Realisierung neuer Workflow-Interaktionsformen für Produktentwicklungsprozesse

Diplomarbeit an der Universität Ulm
Fakultät für Informatik

vorgelegt von:

Ralf Sauter

1. Gutachter: Prof. Dr. Peter Dadam
2. Gutachter: Prof. Dr. Michael Weber

Januar 1999

Vorwort

Die vorliegende Diplomarbeit verfolgte das Ziel, das speziell für Produktentwicklungs-prozesse entworfene WEP-Workflow-Management-System auf der Basis einer Workflow-Beschreibungssprache weiterzuentwickeln, anhand eines Beispiels die Zweckmäßigkeit zu überprüfen und die Mechanismen in Form einer Workflow-Engine zu implementieren. Zusätzlich wurden die Anforderungen entwickelt, die für die Unterstützung von Produktentwicklungsprozessen erfüllt werden müssen. Andere verwandte Ansätze wurden betrachtet und gegen das WEP-Modell und die Anforderungen abgegrenzt.

Die Arbeit entstand am Forschungsinstitut Ulm der DaimlerChrysler AG. Mein besonderer Dank gilt hier Herrn Thomas Beuter für die sowohl fachlich, als auch menschlich hervorragende Betreuung, für die wertvollen Tips und Anregungen und für die Durchsicht und Korrektur meiner Ausarbeitung. Spezieller Dank gilt dem unermüdlichen Engagement, mir die Kommaregeln beizubringen. Weiterer Dank gebührt Herrn Bernd Hassar, der mir am Anfang einige Fragen zu Metaphase beantworten mußte, und den Herren Prof. Dr. Peter Dadam und Prof. Dr. Michael Weber für die Betreuung der Arbeit.

Ulm, im Januar 1999 Ralf Sauter

Inhaltsverzeichnis

1 Einleitung

Betrachtet man die wachsende Komplexität technischer Produkte und Entwicklungen, so stellt man fest, daß der Entwicklungsprozeß typischerweise nicht mehr von einer einzelnen Person durchgeführt wird. An die Stelle des einzelnen Entwicklers, der über alles informiert ist und volle Kontrolle über die Arbeit hat, tritt eine Gruppe von Entwicklern. Jeder Entwickler dieser Gruppe bearbeitet dabei nur einen Teilprozeß zur Lösung der gesamten Aufgabe.

Abbildung 1.1 verdeutlicht den Unterschied zwischen der Arbeit einer einzelnen Person und der Zusammenarbeit mehrerer in einer Gruppe. Als einzelner führt man alle Arbeiten hintereinander aus. Die Gesamtaufgabe in Teile zu zerlegen, hat hier keinen praktischen Nutzen. Arbeiten mehrere Personen zusammen, kann jeder eine Teilaufgabe übernehmen. So lassen sich viele Aufgabe gleichzeitig erledigen und damit die Ausführungszeit der Gesamtaufgabe verkürzen.

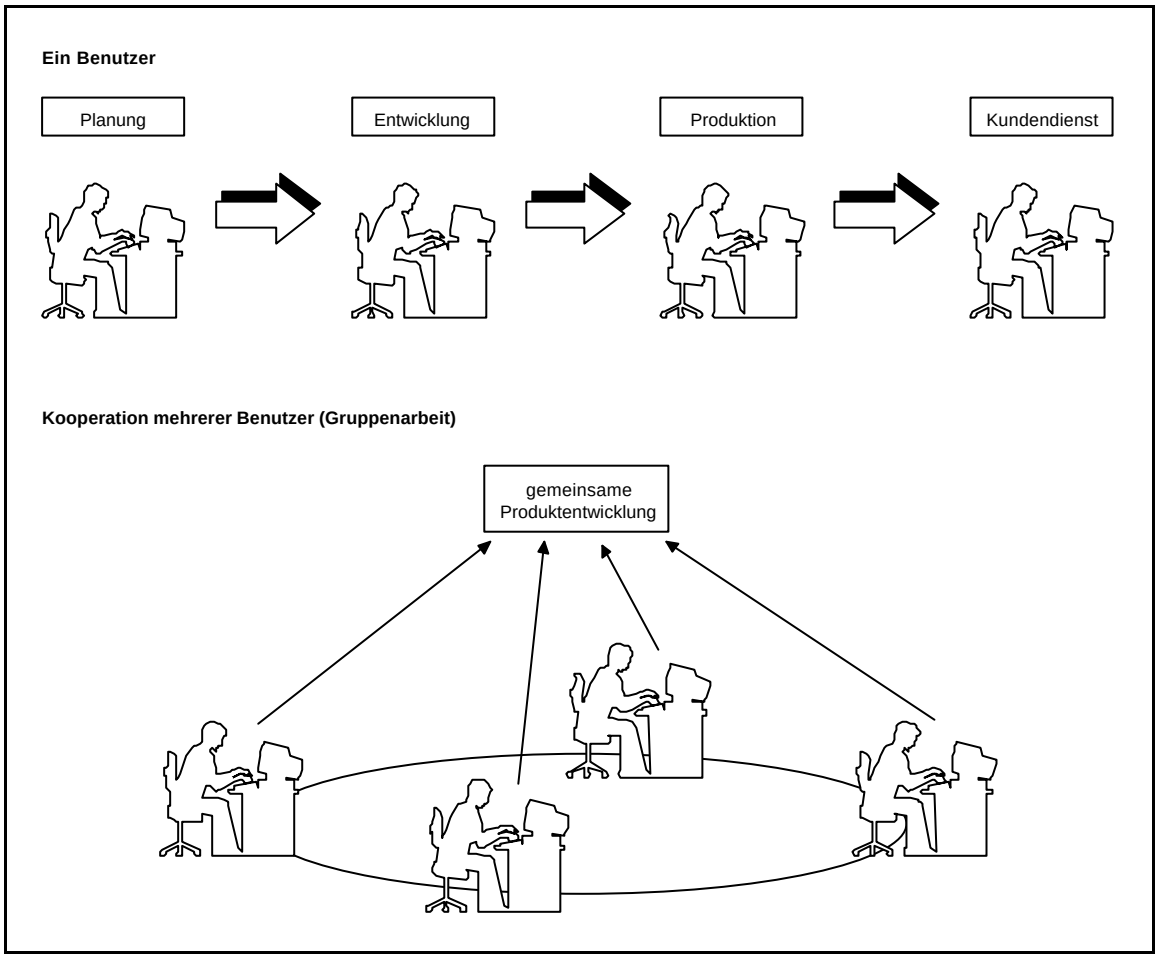


Abbildung 1.1: Gegenüberstellung von einem Benutzer und der Kooperation mehrerer Benutzer

Für die Kontrolle und Unterstützung der einzelnen Entwickler in der Gruppenarbeit sind neue Methoden erforderlich. In der Vor-Computer-Ära mußten diese Aufgaben von anderen Personen übernommen werden. Diese hatten die Gesamtaufgabe so zu strukturieren, daß jeder Entwickler seine Aufgabe möglichst unabhängig von den anderen Entwicklern bearbeiten konnte. War dies nicht möglich, so wurden weitere Mitarbeiter nötig, die für die Unterstützung und Koordination der unterschiedlichen Teilprozesse verantwortlich waren. Auch sind gemeinsame Treffen der Entwickler denkbar, in denen dann an der Verhandlung über

individuelle Entwicklungsziele und dem Austausch von Teilergebnissen gearbeitet wird. Dies führt jedoch zu erheblichen Problemen und Mehraufwand, sollten die Entwickler, wie heute immer mehr üblich, an unterschiedlichen Orten arbeiten.

1.1 Motivation

Inzwischen bietet die computerunterstützte Arbeit verschiedene Ansatzpunkte zur Unterstützung und Kontrolle der Mitarbeiter in der Gruppenarbeit. Klassische Informationssysteme, wie datenbankbasierte Anwendungen, eignen sich jedoch nur sehr unzureichend. Sie wurden zur statischen Verwaltung von Informationen entwickelt. So widerspricht denn auch das ACID-Paradigma[1] (vgl. [LoSc93]), das für konventionelle Transaktionen definiert wurde, teilweise den Anforderungen an die Unterstützung und Kooperation einer Gruppe von Entwicklern. Serialisierbarkeit als der Begriff der Korrektheit ist hier zu restriktiv. So steht beispielsweise die Einbenutzersicht (Isolation), die für eine Trennung zwischen gleichzeitig ablaufenden Transaktionen sorgt, der Kooperation mehrerer Bearbeiter geradezu entgegen. Die Ununterbrechbarkeit (Atomicity) einer Transaktion weiterhin, ist nicht ausreichend für die langen Ausführungszeiten von Entwicklungs-werkzeugen, wie sie heute von Entwicklern eingesetzt werden. Eine erweiterte Definition des Transaktionsbegriffes, wie zum Beispiel verschachtelten Transaktionen (vgl. [GrRe93]), können zwar durchaus zu einer besseren Lösung beitragen, das Problem an sich aber auch nicht lösen.

Das Hauptproblem ist der dynamische Teil der Entwicklungsprozesse. Darunter fällt beispielsweise das Zuteilen neuer Aufträge, der Informations- und Datenaustausch zwischen Entwicklern oder die Festlegung und Überwachung zeitlicher Beschränkungen und Fristen. Diese Arbeiten erfolgen häufig informell zwischen den beteiligten Personen und verursachen einen nicht unerheblichen Zusatzaufwand. In Zukunft wird dies dennoch immer wichtiger. Personen mit unterschiedlichem Wissenshintergrund und unterschiedlicher Ausbildung arbeiten an verschiedenen, zum Teil weit auseinanderliegenden Orten zusammen, um das gesteckte Aufgabenziel zu erfüllen.

Eine EDV-Unterstützung kann diesen Zusatzaufwand, der im wesentlichen durch die Koordination der verteilten Arbeitsabläufe entsteht, entsprechend reduzieren. Eine Lösung für die dynamischen Abläufe mit herkömmlichen Programmiermethoden führt jedoch zu sehr komplexen Anwendungsprogrammen. Der gesamte Kontrollfluß muß explizit ausprogrammiert werden. Dazu kommt, daß sich diese programmierten (‘hart verdrahteten’) Abläufe dann nur noch schwer kontrollieren, warten und an Änderungen anpassen lassen.

Verschiedene Ansätze und Ideen aus dem Forschungsgebiet der computerunterstützten Gruppenarbeit (CSCW - Computer Supported Cooperative Work) erlauben eine elegantere Unterstützung dynamischer Abläufe. Dies betrifft insbesondere die Bereiche des Workflow-Managements, des Projekt-Managements und der Groupware. Im folgenden werden wir diese Bereiche etwas genauer betrachten.

[1] ACID = **A**tomicity, **C**onsistency, **I**solation, **D**urability

1.2 Begriffe der Gruppenarbeit

Der Oberbegriff **CSCW (Computer Supported Cooperative Work)** umfaßt ein großes Gebiet von aktuellen Forschungsbereichen. In ihm lassen sich prinzipiell alle Sparten aus Hardware und Software zusammenfassen, die irgendwie die Zusammenarbeit mehrerer Personen unterstützen. Dabei ist es nebensächlich, ob diese Personen gemeinsam komplexe Daten bearbeiten, oder ob sie einfach nur per Computer Nachrichten austauschen. Ebenso ist es grundsätzlich unbedeutend, ob die Personen durch mehrere tausend Kilometer getrennt sind, oder ob sie sich einen Computer an einem Ort teilen.

Unter CSCW wird dabei ganz abstrakt ein interdisziplinäres Forschungsgebiet aus Informatik, Soziologie, Psychologie, Arbeits- und Organisationswissenschaften, Anthropologie, Ethnographie, Wirtschaftsinformatik, Wirtschaftswissenschaften, u.a. verstanden, das sich mit Gruppenarbeit und die Gruppenarbeit unterstützender Informations- und Kommunikationstechnologie befaßt. Wobei der Terminus CSCW eigentlich eher als Schlagwort anzusehen ist, und ihm keine weitere Bedeutung zukommt (vgl. [HKS94]). Eine etwas umgangssprachlichere Beschreibung stammt aus [EGR91]: Die Forschung um CSCW beobachtet, wie Menschen im Team arbeiten und versucht herauszufinden, wie die Technik, speziell Computersysteme, zur Lösung der Aufgabe beitragen können.

Die Hauptaspekte, die CSCW umfaßt, und die hier nur beispielhaft erwähnt werden, sind

- die Art und Weise der Zusammenarbeit von Menschen innerhalb von Gruppen,
- die Koordination der Arbeit,
- die Anforderungen an die angewandten Technologien und deren Auswirkung auf den Menschen und
- die Auswirkungen auf und Anforderungen an die Organisationsstrukturen.

Um zu entscheiden, wie die Interaktion zwischen mehreren Personen in einer Gruppe am besten unterstützt werden kann, werden drei Schlüsselbegriffe (Eckpunkte der Gruppenarbeit, vgl. [EGR91]) eingeführt:

- Communication (Kommunikation)
- Collaboration, Cooperation (Kooperation, Zusammenarbeit)
- Coordination (Koordination)

Die computerbasierte beziehungsweise computervermittelte **Kommunikation** kann dabei in die synchrone Kommunikation (z.B. Telefon) und die asynchrone Kommunikation (z.B. E-Mail) aufgeteilt werden. Zwischen diesen zwei Kommunikationsformen gibt es trotz des Fortschritts der Computertechnik und der Telekommunikation immer noch gewisse Lücken. So ist es weiterhin mit gewissem Aufwand verbunden oder wird mit schlechter Qualität bestraft, wenn man versucht über eine Telefonverbindung ein Textdokument auszutauschen. Ebenso funktionieren Echtzeitgespräche über Computer noch bei weitem nicht einwandfrei. Diese Probleme werden sich jedoch durch die Weiterentwicklungen der Konferenz-Systeme beziehungsweise Übertragungsprotokolle und einen großzügigeren Ausbau der Datenverbindungen auf Dauer beheben lassen.

Der zweite Eckpunkt ist die **Kooperation**. Sie bedingt das Teilen von gemeinsamen Informationen. Das heißt, daß sich jeder einzelne Benutzer eines CSCW-Systems bewußt sein muß, daß auch andere an der von ihm bearbeiteten Aufgabe beteiligt sind. Dies ist das genaue Gegenteil zu dem Ansatz den ein Datenbank-System verfolgt. Hier versucht man mittels Transaktionskonzepten die einzelnen Benutzer möglichst zu isolieren und ihnen ein Ein-Benutzer-System vorzuspielen (Isolation-Prinzip). Was hier für CSCW benötigt wird, ist eine gemeinsame

Umgebung, die Informationen über den Gruppenzustand und zu den Aktivitäten jedes Benutzers liefern kann, sofern sie benötigt werden.

Der dritte Punkt ist die **Koordination**. Durch eine geeignete Koordinierung der Gruppenaktivitäten kann die Effektivität der ersten zwei Eckpunkte, der Kommunikation und der Zusammenarbeit, erhöht werden. Dies ist zwar ein gewisser Overhead, der sich aber im Normalfall durch die Lösung von Konflikten und die Vermeidung von wiederholt ausgeführten oder sich widersprechenden Arbeiten bezahlt macht.

CSCW ist also ganz allgemein ein Forschungsfeld, das sich mit der Rolle der Informations- und Kommunikationstechnologie im Rahmen kooperativer Arbeit beschäftigt. Die im folgenden beschriebenen Begriffe der Groupware, des Workflow-Managements und des Projekt-Managements können als Teilbereiche des CSCW betrachtet werden.

1.2.1 Groupware

Die Definition von Groupware ist nicht ganz einheitlich. Groupware wird manchmal einfach mit CSCW gleichgesetzt. So definiert [EGR91]: Groupware ist ein computerbasiertes System, das eine Gruppe von Personen bei einer gemeinsamen Aufgabe unterstützt und dabei eine Schnittstelle zu einer gemeinsamen Arbeitsumgebung bereitstellt. Dies läßt sich auch ganz brauchbar als Definition für CSCW verwenden.

Statt dessen ist Groupware aber wohl eher eine vom Forschungsgebiet der CSCW eingesetzte Technik. Groupware ist eine flexible informations- und kommunikationstechnologische Unterstützung einer kleineren Gruppe, die in Eigenregie unterschiedliche und überwiegend unstrukturierte Aufgaben bearbeitet (vgl. [HKS94]). Die Kernaussage dieser Definition ist sicher die Bearbeitung unterschiedlicher Aufgaben in Eigenregie. Hier kann Groupware also auf die informations- und kommunikationstechnologische Unterstützung reduziert werden. Eine Strukturierung und Kontrolle des Arbeitsablaufs eines einzelnen Mitglieds der Gruppe entfällt, unterstützt wird die Kommunikation, Kooperation und Koordination in Bezug auf die Zusammenarbeit zwischen den Gruppenmitgliedern.

Workgroup Computing ist dabei die Art von Arbeit, die durch Groupware ermöglicht wird. Sie veranlaßt einen Rollenwechsel bezüglich der Verwendung eines Rechnersystems: vom passiven Problemlöser, zum aktiven Medium, das die Interaktion zwischen menschlichen Benutzern unterstützen und kontrollieren soll (vgl. [Jabl95a]).

Das Ziel der Groupware ist es also, eine Gruppe in der internen Kommunikation, der Kooperation und der Koordination des gemeinsamen Arbeitsablaufs zu unterstützen.

1.2.2 Workflow-Management

Bei Workflow-Management steht die Kommunikation eher im Hintergrund. Der Kern des Workflow-Managements ist die Koordination und Kooperation strukturierter Arbeitsabläufe. Dabei bietet die hier grundsätzliche Trennung von Ablauflogik (dynamisch) und dem eigentlichen Anwendungscode (statisch) einen vielversprechenden Ansatz.

Die zentrale Rolle für die Strukturierung von Arbeitsvorgängen eines Unternehmens ist der Geschäftsprozeß. Er faßt eine Menge von Aktivitäten zusammen, die durch ein gemeinsames Geschäftsziel verbunden sind. Geschäftsprozesse können meist automatisiert und damit elektronisch verarbeitet werden. Diese Darstellung wird dann als Arbeitsfluß (Workflow) bezeichnet. Ein Workflow-Management-System überwacht die Ausführung solcher Workflows.

Laut [Frau98] eignen sich Workflow-Management-Systeme insbesondere zur Unterstützung von stark strukturierten Prozessen, das heißt Prozessen, die

- eine Reihe von Aktivitäten in einer bestimmten Reihenfolge oder parallel zueinander umfassen,
- immer wieder in der gleichen oder einer ähnlichen Form auftreten,
- mehrere Personen involvieren und
- einem starken Koordinierungsbedarf unterliegen.

In diesem Zusammenhang spricht man dann auch von prozeßorientierten Systemen. Konventionelle Workflow-Management-Systeme sind demnach Systeme, die zum Charakteristikum haben, Prozeßabläufe nach einem vorher definierten Modell zu steuern und eignen sich dadurch besonders für stark strukturierte und arbeitsteilige Organisationen. Aus diesen Gründen könnte man Workflow-Management-Systeme auch als spezielle Groupware-Systeme bezeichnen (vgl. [Jabl95a]), bei denen die Eckpunkte Kooperation und Koordination speziell auf einen vordefinierten Ablaufplan ausgerichtet sind. Workflow-Management-Systeme sind eine aktive Software, das heißt, der Computer übernimmt von sich aus einen aktiven Part in der Steuerung und Überwachung des zu erreichenden Aufgabenziels.

Von Workflow-Management-Systemen bearbeitete Prozesse (Workflows) werden charakterisiert (vgl. [Jabl95b]) durch

- mehrere an der Ausführung beteiligte Personen (Arbeitsteiligkeit),
- mehrere Teilaufgaben,
- Datenverwaltung und Datentransfer,
- Koordination,
- vordefinierte Ablaufstrukturen und
- Fehlerbehandlung.

Die **Workflow Management Coalition** (WfMC) beschreibt ein Workflow-Management-System so: Ein System, das mittels Software Workflows definiert, erstellt und ausführt. Die Software läuft auf einer oder mehreren Workflow-Engines, die in der Lage sind die Prozeßdefinition zu interpretieren, mit Workflow-Teilnehmern zu kommunizieren und, wenn erforderlich, Anwendungsprogramme und Hilfsprogramme zu starten. Ein Workflow ist dabei die Automatisierung eines Geschäftsprozesses, also ein vormodellierter Ablaufplan (vgl. [WfMC96]).

Das in Abbildung 1.2 dargestellte Basismodell der WfMC soll einerseits die wesentlichen Charakteristiken eines Workflow-Management-Systems zum Ausdruck bringen und andererseits die Beziehungen zwischen deren Funktionen verdeutlichen. Dabei wählte man jedoch eine so abstrakte Form, daß sich praktisch jedes erdenkliche Workflow-Management-System darin wiederfinden kann (vgl. [WfMC94], [Vers95]). Gut deutlich wird jedoch die Trennung von Build-Time-Komponente und Run-Time-Komponente.

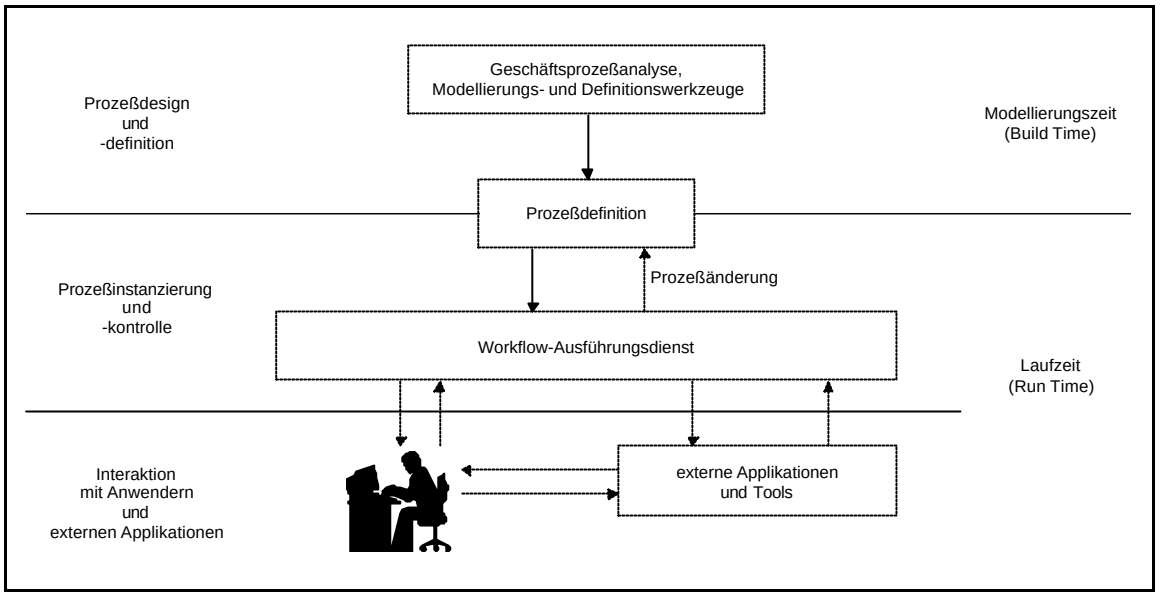


Abbildung 1.2: Basismodell der Workflow Management Coalition

Die Workflow Management Coalition ist eine non-profit Organisation, der inzwischen die meisten Hersteller von Workflow-Management-Systemen angehören. Ihr erklärtes Ziel sind Standardisierungsmaßnahmen im Bereich Workflow-Management. Ihr Themenschwerpunkt ist die Ausarbeitung der Schnittstellen zwischen den im Referenzmodell (Abbildung 1.3) identifizierten Teilkomponenten eines Workflow-Management-Systems. Bis dato geht es weniger um die internen Definitionen, weshalb wir die Ansätze der Workflow Management Coalition hier nicht weiterverfolgen wollen.

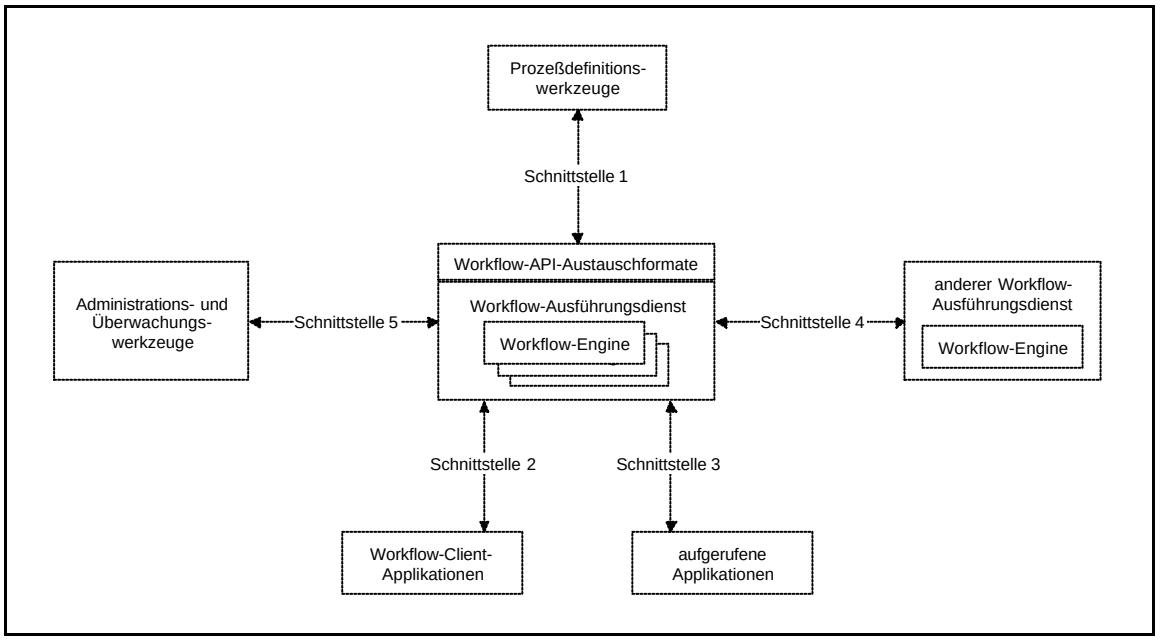


Abbildung 1.3: Referenzmodell der Workflow Management Coalition

Durch die Festlegung auf vormodellierte und stark strukturierte Prozesse und der damit verbundenen geringen Flexibilität ist der praktische Einsatz von Standard-Workflow-Management-Systemen derzeit auf ein relativ kleines Spektrum von Anwendungen beschränkt. Ein weiterer Punkt, der die flexible Anwendung unterbindet, ist die normalerweise vorhandene

Unterscheidung von Modellierungs- und Laufzeitkomponente (vgl. [Joer98]). Die Vormodellierung des Workflows geschieht in der Modellierungskomponente (Build-time). Die Laufzeitkomponente (Run-time) kontrolliert den Ablauf des fertigen Workflows. Diese Trennung eines Workflow-Management-Systems in zwei getrennte Komponenten unterbindet Änderungen an einem Workflow, der bereits ausgeführt wird. Muß der Workflow an neue Gegebenheiten angepaßt werden, so muß er aus der Ausführung und der Kontrolle der Laufzeitkomponente herausgenommen werden und in der Modellierungskomponente überarbeitet werden. Erst danach kann er wieder gestartet werden.
Die Neumodellierung über die Build-time-Komponente ist allerdings meist mit ziemlichem Aufwand verbunden. Daraus läßt sich schließen, daß Workflow-Management-Systeme nur für Aufgabenbereiche geeignet sind, deren Ablauflogik sich fest spezifizieren läßt. Seine Vorteile liegen also bei Aufgaben, an denen wenig geändert wird und die sich ohne Änderungen oft wiederholen.

Durch die zur Laufzeit wenig anpassungsfähige Prozeßabarbeitung, erfordert ein Workflow-Management-System auf jeden Fall eine Gesamtbetrachtung der Arbeitsabläufe in einem Unternehmen und sollte sinnvollerweise erst nach einem gründlichen Reengineering der Aufbau- und Ablauforganisation installiert und verwendet werden. Für ein umfassendes, unternehmensweites Einsatzfeld müssen Workflow-Management-Systeme folgende drei Merkmale unterstützen (vgl. [Jabl95a]):

- **Skalierbarkeit**: Es ist zu erwarten, daß sowohl die Zahl an WfMS-Benutzern, als auch die Zahl auszuführender Workflows ständig steigen wird.
- **Integration** bestehender Software: Trotz Restrukturierung eines Anwendungssystems für den Einsatz des WfMS wird die weitere Verwendung bestehender Software unabdingbar sein.
- **Transparenz**: WfMS werden in verteilten und heterogenen Hard- und Softwareumgebungen eingesetzt. Dies sollte dem Benutzer jedoch verborgen bleiben.

1.2.3 Projekt-Management

Konventionelle Workflow-Management-Systeme beruhen auf vormodellierten und stark strukturierten Prozessen. Sie benötigen also ein vollständiges Prozeßmodell vor Beginn der Ausführung. Da der Aufwand der Entwicklung eines Prozeßmodells sehr groß ist, werden sie gewöhnlich nur für sich wiederholende Prozesse eingesetzt. So ist es meist nicht durchführbar, sie für die Projektplanung, die für jedes Projekt neu durchgeführt wird, einzusetzen. Projekt-Management-Systeme versuchen mittels Zeitmanagement eine Struktur in den Prozeßablauf einzubringen. Bei ihnen ist der Prozeßablauf nicht vormodelliert, sondern wird durch die Anpassung und Einbeziehung von Ressourcen, also zum Beispiel des vorhandenen Personals, der Hilfsmittel und der zeitlichen Gegebenheiten, erstellt.

Besonders wichtig ist hier die Koordination, um verschiedene Teilaufgaben so zuzuordnen, daß das Projekt mit möglichst wenig Ressourcen und Aufwand und in möglichst kurzer Zeit realisiert werden kann. Laut [Burg95] ist die zentrale Aufgabe einer zielgerichteten Projektplanung und Projektsteuerung, die sachgerechte, termin- und aufwandsgerechte sowie kostengerechte Abwicklung eines Projekts. Um dies zu erreichen, muß das Projekt-Management in vielfältiger Weise auf den Projektablauf ‘regelnd’ einwirken. Einerseits werden für das Projekt Planvorgaben gemacht, auf deren Basis steuernde Maßnahmen auf den Ablauf einwirken;

andererseits müssen an definierten Stellen des Prozeßablaufs bewertende Meßgrößen zur Beurteilung ermittelt und ausgewertet werden. Vorraussetzung hierfür ist allerdings immer, daß das Projekt in für das Projekt-Management überschaubare und damit hantierbare Portionen zerlegt wird.

Die Tätigkeiten des Projekt-Managements (vgl. [Kupp93]) lassen sich sehr gut auflisten. Es muß

- planen: Personal, Tätigkeiten, Ressourcen, Termine, Tests, Dokumentation, Wartung, ...
- organisieren: Zuordnung von Tätigkeiten zu Personen, Hilfsmitteln, ...
- schätzen: Zeitdauer und Aufwand
- kontrollieren: Vergleich des tatsächlichen mit dem geplanten Zustand
- steuern: Korrekturmaßnahmen einleiten
- koordinieren: alle beteiligten Funktionen und Bereiche
- informieren: nach oben zum Management, sowie nach unten in die Projektgruppen

Bei konventionellen Systemen läßt sich der Ablauf eines Projekts auf vier Phasen festlegen (vgl. [Jung97] und [Burg95]):

1) **Projektdefinition**: Legt die Grundlagen des Projekts fest und umfaßt die Gründung des Projekts, Definition des Projektziels, Organisation des Projekts und Organisation des Prozesses.

2) **Projektplanung**: Gibt die Rahmendaten des Projekts vor und umfaßt die Strukturplanung, Aufwandsschätzung, Arbeitsplanung und Kostenplanung.

3) **Projektkontrolle**: Soll frühzeitig Planabweichungen aufzeigen und umfaßt die Terminkontrolle, Aufwands- und Kostenkontrolle, Sachfortschrittskontrolle, Qualitätssicherung und Projektdokumentation.

4) **Projektabschluß**: Sichert das korrekte Projektende und umfaßt die Produktabnahme, Projektabschlußanalyse, Erfahrungssicherung und Projektauflösung.

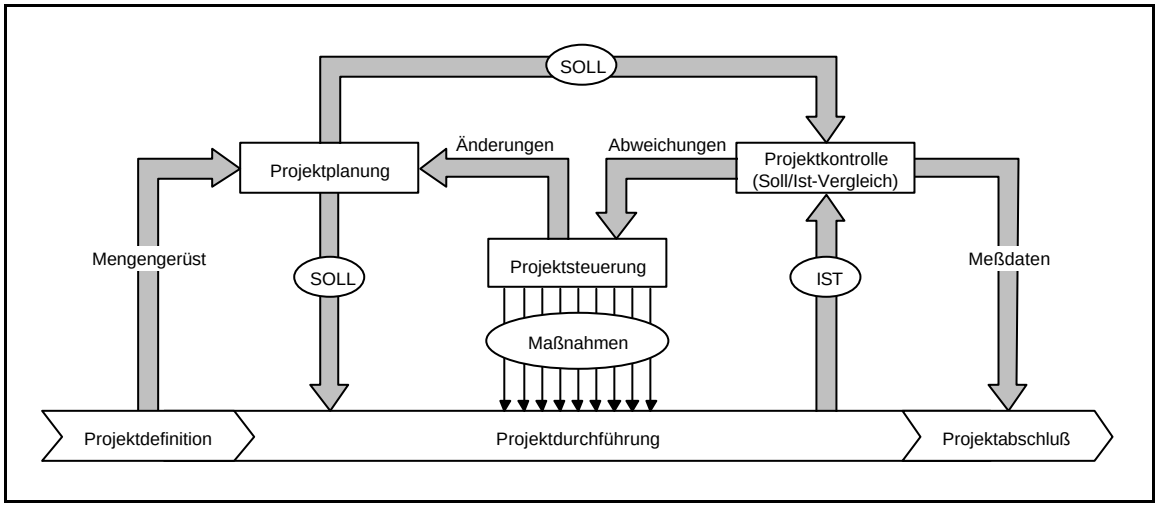


Abbildung 1.4: Projekt-Management Regelkreis

Die vorgenannten Aufgabenbereiche des Projektmanagements lassen sich in ihrem Zusammenwirken zur Projektsteuerung und Projektdurchführung als Regelkreis darstellen (siehe Abbildung 1.4). Wie das Bild zeigt, gibt die Projektplanung auf Basis der Projektdefinition die Planwerte als Führungsgrößen (SOLL) für die Projektdurchführung vor. Durch die Projektkontrolle werden die Meßgrößen (IST) abgefragt und mit den Führungsgrößen

verglichen. Bei Abweichungen sind im Rahmen der Projektsteuerung entweder geeignete Maßnahmen vorzunehmen oder Planvorgaben zu ändern.

1.3 Produktentwicklungsprozesse

So allgemein man den Begriff Projekt auch definieren kann, so unterschiedlich können die einzelnen Projekte sein. Ziel dieser Arbeit ist die spezielle Betrachtung von Produktentwicklungsprozessen, also Entwicklungsprojekten. Diese Art der Projekte haben in der Regel ein klar definiertes Ziel (Entwicklungsziel), nämlich das für die Fertigung freizugebende Produkt. Wegen der meist festumrissenen Planungsbasis sind die Unsicherheiten im Erreichen des Projektziels relativ gering. Bei neuen Produkten sollte der Kreativität der Mitarbeiter und der Ideenfindung dennoch genügend Freiraum gelassen werden, auch wenn sich die Unsicherheiten dadurch wieder erhöhen. Bei Entwicklungs-projekten ist auf das Projektmanagement besonderes Gewicht zu legen, da gerade im Entwicklungsbereich, wegen des marktbestimmenden Einflusses eines früheren Markteintritts, die Durchlaufzeiten verkürzt werden müssen.

Produktentwicklungsprozesse lassen sich charakterisieren durch eine

- zunehmende Konkretisierung von der Idee zum fertigen Produkt und die
- Aufgliederung in Teilaufgaben aufgrund der Komplexität der Entwicklungsaufgabe.

Jede Teilaufgabe besorgt die Ausarbeitung eines Teils des zu entwicklenden Produkts. Dabei werden neben der Festlegung, welche Teile eines Produkts in einer Teilaufgabe konkretisiert werden sollen, Meilensteine definiert, die Vorgaben über den zeitlichen Verlauf der jeweiligen Teilentwicklung spezifizieren.

Das Schema eines Produktentwicklungsprozesses orientiert sich für gewöhnlich an den Organisationsstrukturen eines Unternehmens und sieht somit im großen und ganzen immer gleich aus. Wir wollen uns an der Produktentwicklung im Fahrzeugbereich orientieren, so wie sie nach Erhebungen im Daimler-Benz-Konzern stattfindet. Abbildung 1.5 stellt die äußere vorgegebene Struktur dar. Bei der Entwicklung anderer komplexer Produkte, zum Beispiel in der Unterhaltungsindustrie, sind die Anforderungen jedoch sehr ähnlich.

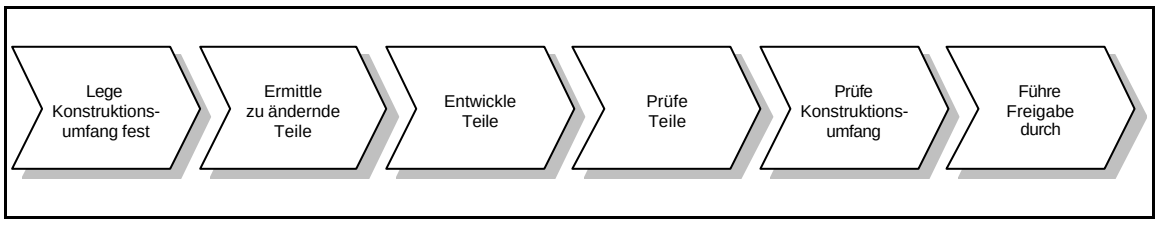


Abbildung 1.5: Prinzipieller Ablauf für einen Produktentwicklungsprozeß

Die Aufgliederung in Teilaufgaben erfolgt hier anhand der zugrundeliegenden Datenstruktur. Sie entsteht übergreifend über den Aufgaben ‘Entwickle Teile’ und ‘Prüfe Teile’. Dabei teilt sich der Prozeßablauf in soviele parallele Zweige, wie das vollständige Produkt sich in einzeln entwickelbare Teile zerlegen läßt. Diese Aufteilung kann sich natürlich auch schrittweise anhand einer hierarchischen Baumstruktur entwickeln.

Wir wollen den in den folgenden Kapiteln immer wieder betrachteten Beispielen eine hierarchische Objektstruktur zugrundelegen, wie sie in Abbildung 1.6 dargestellt wird.

Ausgehend vom Objekt KEM (Konstruktions-Einsatz-Meldung, entspricht einem Konstruktions- / Änderungsauftrag) zerlegt sich die Struktur in variabel viele, vom Auftrag betroffene DMUs (Digital Mock-Up, virtuelles Konstruktionsmodell, entspricht einem Fahrzeugtyp). Ein DMU zerlegt sich in beliebig viele Module (Baugruppen) und diese wiederum in Einzelteile.

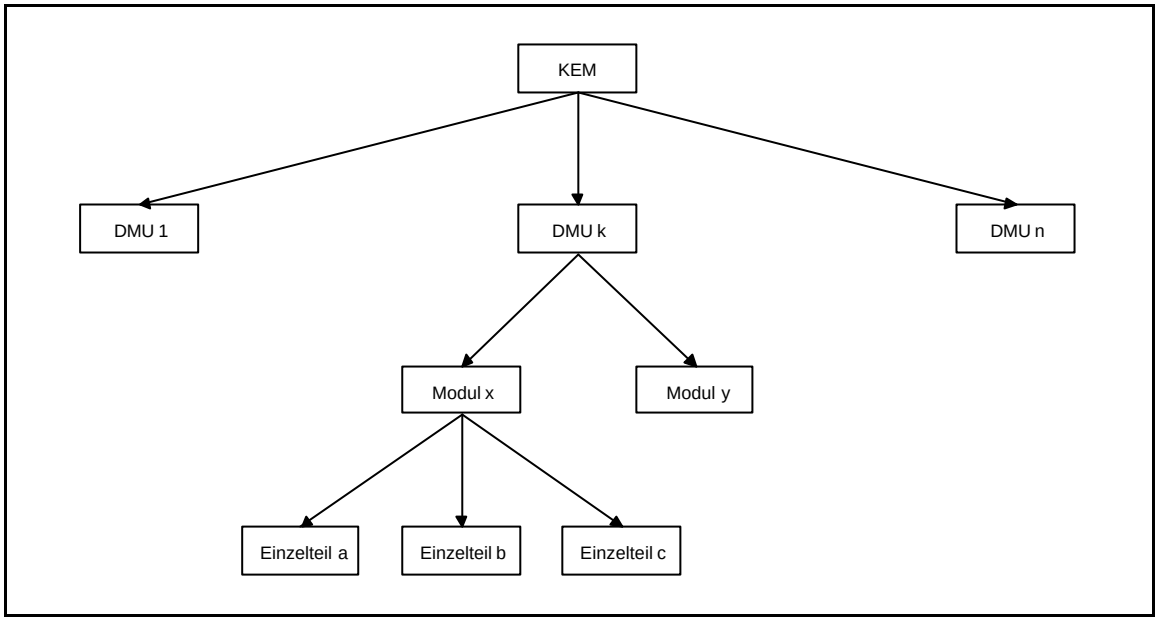


Abbildung 1.6: Beispiel einer Objektstruktur für einen Produktentwicklungsprozeß

Ein solcher Produktentwicklungsprozeß besteht nun aus der äußeren vorgegebenen Struktur (statisch durch die Prozeßstruktur vorgegeben) und der Aufgliederung in variabel viele Teilaufgaben (dynamisch anhand der zugrundeliegenden Datenstruktur).

Durch die bisherigen Betrachtungen lassen sich spezifische Eigenschaften von Produktentwicklungsprozessen festlegen, die ein System möglichst unterstützen sollte, um den Anforderungen zu genügen. Die Eigenschaften sind:

- Anfangs existiert nur eine Grobplanung, die Feinplanung ist erst zur Laufzeit möglich, wenn mehr Details verfügbar werden, beziehungsweise die Teilaufgaben zugeordnet wurden.
- Produktentwicklungsprozesse enthalten in den Entwicklungsaufgaben einen hohen Anteil kreativer Tätigkeiten, die sich nicht oder nur teilweise strukturieren lassen.
- Produktentwicklungsphasen zeichnen sich durch geringe Wiederholungsraten und längere Laufzeiten aus als gewöhnliche betriebliche Prozesse.

1.4 Anforderungen und Zielsetzung

Auf die Anforderungen, die erfüllt werden müssen, um Produktentwicklungsprozesse zu unterstützen, wurde bereits eingegangen. Interessant sind außerdem die Anforderungen, die von Bearbeitern und von der Geschäftsführung in diesem Bezug an das System gestellt werden.

Der Wunsch der Bearbeiter ist eine freie Arbeitsweise, die sie möglichst wenig einschränkt. Problematisch dabei ist die meist sehr starke Strukturierung bezüglich der zu erledigenden

Aufgabe, wie es von bestehenden Workflow- und Projekt-Management-Systemen vorgegeben wird. Eine solchermaßen vorgeschriebene Arbeitsweise verhindert die individuelle und kreative Arbeitsgestaltung. Das heißt, das System soll den Bearbeiter in seiner Aufgabe unterstützen, aber keine Arbeitsweise oder Arbeitsreihenfolge aufzwingen.

Die Ziele der Geschäftsführung sind selbstverständlich etwas anderer Natur. Ihr Augenmerk ist ausgerichtet auf die Zukunft der Firma. Der problemlosen Zukunft entgegen, wirkt eine sich immer weiter verschärfende Wettbewerbssituation, die für Preisverfall und Innovationsdruck sorgt. Um in dieser Wettbewerbssituation besser bestehen zu können, läßt sich der Wunsch der Geschäftsführung auf einen Nenner bringen: Verkürzung der Produktentwicklungszeiten bei gleichzeitiger Verbesserung der Produktqualität und Senkung der Produktionskosten. Durch die Erreichung dieser Ziele ist eine gewisse Strukturierung der Entwicklungsprozesse unabdingbar. Dadurch entsteht natürlich ein Zielkonflikt im Vergleich zu den Zielen der Bearbeiter.

Aus den Wünschen der Benutzer und der Geschäftsführung, sowie aus den Forderungen die für die Unterstützung der speziellen Produktentwicklungsprozesse entstehen, lassen sich nun konkrete Anforderungen formulieren. Diese Anforderungen sollten bei der Auswahl beziehungsweise Entwicklung eines Systems für unsere Zwecke erfüllt werden. Ein solches System sollte also Unterstützung bieten für

- flexible Abläufe innerhalb des Prozesses wegen der Aufgliederung in Teilaufgaben,
- unstrukturierte kreative Teilprozesse zur freien Entfaltung der Bearbeiter während der Entwicklungsphase,
- ein objektorientiertes Datenmodell für die strukturierten Daten der Produktentwicklung,
- simultanes Bearbeiten sequentieller Prozeßschritte zur Verkürzung der Entwicklungszeiten,
- ein Zeitmanagement zur Einhaltung vorgegebener Fristen,
- ein Informationsmanagement, um Bearbeiter an Zeitüberschreitungen und Fristen zu erinnern,
- synchrone Gruppenarbeitsmechanismen zum einfacheren Datenaustausch und zur einfacheren Kommunikation zwischen Bearbeitern,
- einfache Anpassung des Systems an Änderungen, um schneller auf Kundenwünsche und Innovationen reagieren zu können und
- eine einfache Prozeßverfolgung, um Probleme und den Entwicklungsstand schneller erkennen zu können.

Ein System, das manche der aufgeführten Punkte nur unvollständig oder gar nicht unterstützt und damit nicht alle Anforderungen erfüllt, ist jedoch nicht zwangsweise für Produktentwicklungsprozesse ungeeignet.

Groupware-Systeme sind laut unserer Festlegung hauptsächlich für die Kommunikation und die Koordination unstrukturierter Gruppenarbeit angelegt. Eine Unterstützung auf der Ebene einzelner Bearbeiter existiert ebensowenig, wie für eine strukturierte Zusammenarbeit.

Standard-Workflow-Management-Systeme haben generelle Probleme mit flexiblen Abläufen innerhalb eines Produktentwicklungsprozesses, da sie von vorstrukturierten und fertigen Abläufen ausgehen. Auch sind simultane Prozeßschritte bislang so gut wie nicht möglich. Weiterhin enthalten Workflow-Management-Systeme im Regelfall kein Zeitmanagement.

Zeitmanagement ist dagegen ein grundsätzlicher Bestandteil von Standard-Projekt-Management-Systemen. Die Unterstützung flexibler Abläufe jedoch, ist noch problematischer, da bei ihnen gewöhnlich immer strikt zwischen Planung und Ausführung unterschieden wird. Eine spätere Feinplanung während der Laufzeit ist in der Regel nicht möglich. Auch die Unterstützung kreativer Teilprozesse scheitert daran.

Das Hauptproblem aller konventioneller Systeme ist die fehlende Unterstützung zusätzlicher dynamischer und auch kreativer unstrukturierter Abläufe. Auch fortführende Interaktions-

möglichkeiten, wie das simultane Bearbeiten sequentieller Prozeßschritte (Simultaneous Engineering), sind kaum vorhanden.

1.5 Ziel der Diplomarbeit

Die Diplomarbeit verfolgte im wesentlichen zwei Ziele:

- Im konzeptionellen Teil mußte das WEP-Workflow-Management-System auf der Basis einer Workflow-Beschreibungssprache weiterentwickelt werden. Anhand eines zwar vereinfachten aber dennoch realitätsnahen Beispiels war die Zweckmäßigkeit und Anwendung der Mechanismen des WEP-Modells zu überprüfen. Desweiteren wurden mehrere Bereiche der Gruppenarbeit, die für Produktentwicklungsprozesse in Frage kommen, betrachtet. Dabei wurden verschiedene verwandte Ansätze aus den Bereichen des Workflow-Managements, Projekt-Managements und der Groupware genauer durchleuchtet und mit den gestellten Anforderungen verglichen. Mittels des bereits erwähnten Modells wurde die Umsetzung eines Produktentwicklungsprozesses in das jeweilige System getestet.
- Im Implementierungsteil der Arbeit war die Konzipierung und prototypische Implementierung der Serverkomponente des WEP-Workflow-Management-Systems gefordert. Dabei sollte ermittelt werden, wie sich die Besonderheiten von WEP in der Praxis realisieren lassen. Die Architektur für die Serverkomponente mußte basierend auf einem bestehenden Datenhaltungssystem entwickelt und die Workflow-Beschreibungs-sprache an die Erfordernisse einer Implementierung angepaßt werden. Im Normalfall wird diese Workflow-Beschreibung von der Build-time-Komponente geliefert. Außerdem mußte eine Funktionsschnittstelle definiert werden, über die ein Klient den Server ansprechen kann.

1.6 Gliederung

Die vorliegende Arbeit ist folgendermaßen untergliedert:

In Kapitel 2 wird das grundlegende Modell des WEP-Workflow-Management-Systems beschrieben. Dabei werden die Konstrukte des Modells auch speziell anhand der entwickelten Workflow-Beschreibungssprache dargestellt. Weiterhin wird ein Beispiel, zur Überprüfung der Mechanismen des WEP-Modells entwickelt.

In Kapitel 3 werden die ausgewählten verwandten Ansätze untersucht und mit den gestellten Anforderungen verglichen. Dabei wird auch jeweils versucht, das bereits erwähnte Beispiel möglichst sinnvoll in das betrachtete Modell umzusetzen.

Wichtige Teile der Implementierung werden in Kapitel 4 beschrieben. Darunter fallen vor allem die Architektur der Prototyp-Implementierung und die Beschreibung der Funktionsschnittstelle.

Kapitel 5 gibt eine Zusammenfassung über die Ergebnisse der Arbeit und einen Ausblick auf weiterführende Themen.

Im Anhang wird die in Kapitel 2 entwickelte Workflow-Beschreibungssprache nochmal in der Form von Syntaxdiagrammen dargestellt.

2 Das WEP-Modell

Betrachtet man heutige Workflow-Management- oder Projekt-Management-Technologien, so wird durch die Anforderungen (siehe Kapitel 1) bereits deutlich, daß Produktentwicklungsprozesse nicht entsprechend unterstützt werden können. Das Ziel des im folgenden vorgestellten **WEP**-Modells (**W**orkflow-Management for **E**ngineering **P**rocesses) ist die aktive Unterstützung solcher Entwicklungsprozesse (vgl. [BDS98]). Die Grundidee dabei ist die Erweiterung eines prozeßorientierten Modells mit zielorientierten Aktivitäten. Die Ziele dieser Aktivitäten werden definiert durch die Angabe, wann welche Ausgabeobjekte zur Verfügung stehen müssen. Durch das zusätzliche Einbeziehen von Datenqualitäten wird die vorzeitige Datenweitergabe möglich.

Als Grundlage für WEP und für ein fundiertes Verständnis der Arbeitsweise im Entwicklungsbereich wurden verschiedene Prozeßanalysen in den Entwicklungsbereichen PKW / LKW des Daimler-Benz Konzerns durchgeführt und daraus Anforderungen an eine geeignete Systemumgebung zur Unterstützung von Entwicklungsprozessen abgeleitet. Hierbei wurde festgestellt, daß die am Prozeß beteiligten Personen immer nach dem gleichen Schema zusammenarbeiten. Wiederholende Zusammenarbeit läßt sich also prinzipiell durch Workflow-Management-Technologie sinnvoll unterstützen. Andere Arbeiten bezüglich des Einsatzes von Workflow-Management-Systemen bestätigen dieses Ergebnis (vgl. [Herb96]).

Um den speziellen Anforderungen für Produktentwicklungsprozesse gerecht zu werden, erweitert das WEP-Modell bisherige Workflow-Management-Ansätze durch die Unterstützung variabler Parallelität, schwach strukturierter Teilprozesse, Simultaneous-Engineering-Phasen und der Bereitstellung eines komplexen Datenmodells.

Wenn im folgenden die Konstrukte des WEP-Modells beschrieben werden, so wird dabei zur detaillierteren Erklärung immer die Syntax der Workflow-Beschreibungssprache in leicht vereinfachter Form herangezogen. Für die Erklärung wird die blockorientierte schriftliche Darstellung gewählt, da sie recht einfach Zeile für Zeile beschrieben werden kann. Eine grafische Darstellung, in Form von Syntaxdiagrammen, kann im Anhang nachgeschlagen werden.

2.1 Grundlagen des Modells

Die Grundlage von WEP ist ein prozeßorientiertes Workflow-Modell. Dadurch wird die grobe Struktur eines Entwicklungsprozesses festgelegt und die Führung der am Prozeß beteiligten Mitarbeiter entsprechend der Unternehmensrichtlinien ermöglicht. Die Grobplanung eines WEP-Workflows orientiert sich so direkt an der Organisationsstruktur des Unternehmens.

Bei der Unterstützung schwach strukturierter Teilprozesse wird jedoch auf die aufwendige und wenig nützliche Modellierung der Reihenfolge der Subprozesse verzichtet. Stattdessen werden die Subprozesse eines schwach strukturierten Prozesses zu einer möglicherweise langdauernden zielorientierten Aktivität zusammengefaßt (Kapitel 2.2). Die Ziele einer solchen WEP-Aktivität werden auf der Basis der zu erstellenden Ausgabedaten festgelegt. Dafür wird auf ein objektorientiertes Datenmodell zurückgegriffen, das zur Beschreibung der Datenobjekte dient (Kapitel 2.3).

Die Modellierung eines WEP-Workflows zeigt auf, daß sich zielorientierte Aktivitäten genauso wie herkömmliche Aktivitäten mittels eines Kontrollflusses darstellen lassen (Kapitel 2.4). Das spezielle Konstrukt der Traversierung sorgt dabei für die Unterstützung einer variablen Anzahl paralleler Kontrollflüsse. Dadurch läßt sich die Feinplanung realisieren, die sich an der dem Workflow zugrundeliegenden Datenstruktur orientiert. Für die korrekte Einbeziehung der Datenstruktur in den Kontrollfluß der zielorientierten Aktivitäten sorgt die Datenflußmodellierung (Kapitel 2.5).

Ein Workflow-Management-System, das speziell Entwicklungsprozesse unterstützen will, muß natürlich auch dafür sorgen, daß die vorgegebenen Fristen einer Entwicklung eingehalten werden. Dazu wird ein Zeitmanagement benötigt (Kapitel 2.6). Eine Rollenzuordnung übernimmt die Zuweisung zwischen Aktivitäten und Bearbeitern (Kapitel 2.7).

Die bisherigen Kapitel 2.2 bis 2.7 lassen sich zum Großteil der Workflow-Modellierung zuordnen. Auch ein WEP-Workflow-Management-System besteht grundlegend aus den zwei Komponenten Modellierungsteil (Build-time) und Laufzeitumgebung (Run-time). Die Modellierungskomponente erstellt den Workflow und kann ihn anhand der Modellierungskonstrukte auf die Möglichkeit einer korrekten Ausführung überprüfen. Die Laufzeitkomponente kontrolliert die Abarbeitung des dann fertigen Workflows. Dabei muß sie ständig die Ausführung der Aktivitäten und die Einhaltung vorgegebener Zeiten überwachen. Das Vorgehen der Laufzeitkomponente im Zusammenspiel mit den Benutzerinteraktionen wird in Kapitel 2.8 aufgezeigt.

2.2 Zielorientierte Aktivitäten

Ein wichtiges Konzept des WEP-Modells ist die Integration schwach strukturierter Teilprozesse. Hier stößt eine bei heutigen Workflow-Management-Systemen übliche prozeßorientierte Modellierung von Entwicklungsprozessen an ihre Grenzen. Stattdessen bietet sich eine zieldatenorientierte Modellierungsmethodik an, bei der nicht die Reihenfolge der einzelnen Prozeßschritte, sondern die zu erzeugenden Ergebnisdaten inklusive der zur Verfügung stehenden Zeiträume festgelegt werden. Die Frage, die bei der Modellierung gestellt wird, ist also nicht mehr: ‘Wie soll es gemacht werden?’, sondern: ‘Was soll gemacht werden?’. Statt dem Weg wird das Ziel beschrieben. Durch eine Integration dieser zielorientierten Aktivitäten in den prozeßorientierten Gesamtkontext erlaubt diese Vorgehensweise den Prozeßbeteiligten (wie beispielsweise Entwicklern) zur Laufzeit genügend Freiraum für eine individuelle und kreative Arbeit, ohne die für den Gesamtprozeß notwendige Kontrolle zu vernachlässigen.

Das WEP-Modell verzichtet bei schwach strukturierten Teilprozessen auf die Modellierung der Reihenfolge der Subprozesse. Die Subprozesse werden dafür zu einer zielorientierten Aktivität zusammengefaßt. Innerhalb dieser Aktivität können sie durch den Bearbeiter in beliebiger Reihenfolge und Häufigkeit aufgerufen werden. Dem Bearbeiter werden nur Angaben gemacht, zu welchen Zeitpunkten er welche Ergebnisobjekte und in welcher Datenqualität abzuliefern hat. Ein solches Prozeßziel nennt man Meilenstein. Es gibt keine Vorschriften, wie er dieses Ziel zu erreichen hat. Für die Angabe der Datenqualitäten ist ein erweitertes Datenmodell nötig (siehe Kapitel 2.3 Objektorientiertes Datenmodell). Dadurch werden jedoch zusätzlich weitergehende Mechanismen, wie die vorzeitige Datenweitergabe, ermöglicht (siehe Kapitel 2.8.2 Vorzeitige Datenweitergabe).

Auch bei zielorientierten Aktivitäten müssen, wie bei herkömmlichen Aktivitäten, die benötigten Ein- und Ausgabeobjekte und Bearbeiter festgelegt werden. Da die WEP-Aktivitäten

weitergehende Möglichkeiten bieten, müssen jedoch für die Ein- und Ausgabeobjekte zusätzliche Aspekte berücksichtigt werden. Abbildung 2.1 zeigt die Beschreibung einer zielorientierten Aktivität in der Syntax der Workflow-Beschreibungs-sprache. Zeile 1 definiert den Namen der Aktivität. In Zeile 3 kann die Aufgabe der Aktivität näher beschrieben werden. Zeile 4 definiert den Rollennamen des Benutzers, der für die Ausführung der Aktivität in Frage kommt (siehe auch Kapitel 2.7 Rollenzuordnung).

```
 1   ACTIVITY <ActivityName>
 2   {
 3      DESCRIPTION <ActivityDescription>
 4      PROCESSED BY <UserRoleName>
 5      IN REQUIRED | OPTIONAL <InputObjectName> : <ObjectClass>
 6      {
 7         QUALITYRANGE <QualityLevel>, ...
 8         CONSISTENCYPOLICY UNDO_REDO | MERGE
 9      }
10      OUT <OutputObjectName> : <ObjectClass>
11      {
12         REQUIRED | OPTIONAL IN QUALITYLEVEL <ActivityQualityLevel>
13         {
14            BASED ON OBJECT QUALITYLEVEL <QualityLevel>
15            EXPRESSION <BooleanExpression>
16            PREREQUISITE <InputObjectName>.<QualityLevel>
17            TIMELIMIT <TimeUnit>
18            CONCURRENTMODE CONSOLIDATION | AUTONOMOUS
19            DISTANCE <Distance>
20         }
21      }
22      PROGRAMSTEP <ProgramStepName>
23      RETURNCODE <ReturnCodeName>
24      {
25         REQUIRED <OutputObjectName>.<ActivityQualityLevel>
26      }
27   }
```

Abbildung 2.1: Syntax für die Beschreibung einer zielorientierten Aktivität

Die Festlegung der Ausgabeobjekte (Abbildung 2.1, Zeilen 10 bis 21) einer WEP-Aktivität muß in Zusammenhang mit den Prozeßzielen (Meilensteinen) der Aktivität gesehen werden. Ein Meilenstein ist also ein Ausgabeobjekt in bestimmter Qualität zu einem bestimmten Zeitpunkt. Für jedes Ausgabeobjekt können mehrere Zeiten festgelegt werden, an denen es jeweils in einer bestimmten (anderen) Qualitätsstufe bereitstehen muß. Die Qualitätsstufe des Ausgabeobjekts wird dabei von den Datenqualitäten der Objekte im Datenmodell abgeleitet. Sie kann jedoch zusätzlich durch die Angabe eines booleschen Ausdrucks verfeinert werden. Ein Ausgabeobjekt einer WEP-Aktivität wird also konkret durch folgende Merkmale spezifiziert:

- **OutputObjectName**: Benennung des Ausgabeobjekts innerhalb der Aktivität (Zeile 10).
- **ObjectClass**: Objektklasse des Ausgabeobjekts (Zeile 10). Wird im Datenmodell definiert.
- **Importance**: Ist die Erfüllung dieses Meilensteins zwingend (REQUIRED) oder zusätzlich (OPTIONAL) möglich (Zeile 12).

- **ActivityQualityLevel**: Benennung einer Qualitätsstufe des Ausgabeobjekts innerhalb der Aktivität (Zeile 12). Entspricht einem Meilenstein. Der durch diese Angabe umfaßte Block (Zeilen 12 bis 20) kann mehrfach für ein Ausgabeobjekt auftreten und definiert dabei jeweils eine neue Qualitätsstufe, das heißt, einen neuen Meilenstein.
- **QualityLevel**: Qualitätsstufe basierend auf der Datenqualität der Objektklasse des Ausgabeobjekts (Zeile 14).
- **Expression**: Boolescher Ausdruck, mit dem die Qualitätsstufe des Ausgabeobjekts detaillierter spezifiziert werden kann, um sich von der Qualitätsstufe der Objektklasse abzuheben (Zeile 15). Diese Zeile kann mehrfach angegeben werden. Die verschiedenen booleschen Ausdrücke werden dann durch logisches Und verbunden.
- **Prerequisite**: Gibt an, welche Eingabeobjekte (**InputObjectName**) in welcher Datenqualität (**QualityLevel**) benötigt werden, damit die Qualitätsstufe des Ausgabeobjekts erreicht werden kann (Zeile 16). Auch diese Zeile kann natürlich mehrmals angegeben werden.
- **TimeLimit**: Zeitpunkt ab dem Start der Aktivität, zu dem das Ausgabeobjekt spätestens zur Verfügung stehen muß (Zeile 17). Für nähere Angaben zum Zeit-Management, siehe Kapitel 2.6.
- **ConcurrentMode**: Beschreibt die Strategie, nach der Änderungen an vorzeitig weitergegebenen Daten behandelt werden (Zeile 18). AUTONOMOUS bedeutet, daß die Aktivität eigenmächtig eine neue offizielle Version des Objekts erstellen kann. Abhängige Folgeaktivitäten werden erst nach der Änderung informiert. Im Modus CONSOLIDATION wird eine Abstimmungsrunde einberufen, in der alle beteiligten Bearbeiter über die neue Objektversion beraten können. Entschieden wird letztenendes aber dennoch vom Bearbeiter dieser Aktivität (näheres dazu in Kapitel 2.8.4 Konsolidierungsphasen).
- **Distance**: Zur Beschränkung der Fehlerfortpflanzung. Legt fest, wie viele Aktivitäten weit sich vorläufige Daten von der aktuellen Aktivität entfernen dürfen (Zeile 19).

Für die Eingabeobjekte (Abbildung 2.1, Zeilen 5 bis 9) einer WEP-Aktivität müssen ebenfalls zusätzliche Merkmale beachtet werden. Da beim WEP-Ansatz Aktivitäten bereits mit vorläufigen Eingabedaten bearbeitet werden können, muß sichergestellt werden, daß die Eingabedaten immer in der erforderlichen Datenqualität zur Verfügung stehen. Die Merkmale eines Eingabeobjekts sind:

- **InputObjectName**: Benennung des Eingabeobjekts innerhalb der Aktivität (Zeile 5).
- **ObjectClass**: Objektklasse des Eingabeobjekts (Zeile 5). Wird im Datenmodell definiert.
- **Importance**: Ist das Eingabeobjekt für die Bearbeitung der Aktivität zwingend erforderlich (REQUIRED) oder kann es zusätzlich (OPTIONAL) hinzugezogen werden (Zeile 5).
- **QualityRange**: Gibt die Datenqualitäten (**QualityLevel**), basierend auf den Qualitätsstufen der Objektklasse an, in denen das Eingabeobjekt mindestens verfügbar sein muß, um die Aktivität zu starten (Zeile 7).

- **ConsistencyPolicy**: Spezifiziert, nach welcher Strategie neu eingetroffene Eingabeobjekte in die bereits laufende Aktivität integriert werden (Zeile 8). Bei UNDO_REDO werden eigene Objektversionen der Aktivität als ungültig markiert und die Aktivität danach erneut zur Bearbeitung angeboten. Im MERGE-Verfahren wird versucht, inkonsistente Objektversionen gegeneinander abzugleichen (näheres dazu in Kapitel 2.8.3 Integrationsphasen).

Zusätzlich wird bei einer zielorientierten Aktivität die Beschreibung der in der Aktivität verfügbaren Werkzeuge (Programmschritte) nötig. Herkömmliche Aktivitäten können mit der Ausführung eines Werkzeuges gleichgesetzt werden. Zielorientierte Aktivitäten stellen dem Bearbeiter jedoch eine variable Anzahl verschiedenartiger Werkzeuge zur Verfügung, die dieser seinen Vorstellungen entsprechend einsetzen kann. Diese Werkzeuge werden bei WEP in Form von Programmschritten beschrieben. Dabei erfolgt weder eine Kontrollfluß- noch eine Datenflußmodellierung, das heißt, der Bearbeiter ist selbst verantwortlich, daß er dem Werkzeug die erforderlichen Datenobjekte bereitstellt. Damit die Programmschritte jedoch nicht mit den falschen Objekten ausgeführt werden, wird definiert, welche Eingabe-objekttypen prinzipiell benötigt werden und welche Ausgabeobjekttypen entstehen. Bei der Aktivitätenbeschreibung werden Programmschritte nur noch über ihren Namen spezifiziert (Abbildung 2.1, Zeile 22). Allgemein für den Workflow werden sie separat definiert, da sie in verschiedenen Aktivitäten enthalten sein können. Dies wird durch Abbildung 2.2 dargestellt. Zeile 1 legt den Namen für den Programmschritt fest. In Zeile 3 kann beschrieben werden, was der Programmschritt ausführt.

```
1   PROGRAMSTEP <ProgramStepName>
2   {
3      DESCRIPTION <ProgramStepDescription>
4      EXECUTE <ProgramCall>
5      INPUT <ObjectClass>, ...
6      OUTPUT <ObjectClass>, ...
7   }
```

Abbildung 2.2: Syntax für die Beschreibung eines Programmschrittes

Die charakteristischen Merkmale für die Definition eines Programmschrittes sind:

- **ProgramStepName**: Benennung des Programmschrittes (Zeile 1). Über diesen Namen wird der Programmschritt innerhalb einer Aktivitätenbeschreibung referenziert.
- **Execute**: Gibt das Kommando (**ProgramCall**) vor, das für die Ausführung des Programmschrittes aufgerufen werden muß (Zeile 4).
- **Input**: Definiert die als Eingabe des Programmschrittes erlaubten Objektklassen (Zeile 5). Werden mehrere Objektklassen angegeben, so benötigt der Programmschritt mehrere Eingaben. Bei Angabe mehrerer Input-Zeilen sind verschiedene Kombinationen als Eingabe erlaubt.
- **Output**: Definiert die als Ausgabe des Programmschrittes erlaubten Objektklassen (Zeile 6). Entsprechend zur Input-Zeile.

Das Ergebnis einer zielorientierten Aktivität wird vom Bearbeiter durch die Festlegung eines Rückgabewertes spezifiziert (Abbildung 2.1, Zeilen 23 bis 26). Deshalb wird für jeden möglichen Rückgabewert definiert, welche Ausgabeobjekte dafür in welcher Qualitätsstufe vorzuliegen haben. Damit läßt sich sicherstellen, daß eine Aktivität nicht ohne die erforderlichen Daten beendet wird. Ein Rückgabewert wird durch folgende Merkmale charakterisiert:

- **ReturnCodeName**: Benennung des Rückgabewertes (Zeile 23). Davon können auch verschiedene Zweige des Kontrollflusses abhängen.
- **OutputObjectName**: Verweis auf das, für diesen Rückgabewert benötigte Ausgabeobjekt (Zeile 25).
- **ActivityQualityLevel**: Verweis auf die benötigte Qualitätsstufe des Ausgabeobjekts (Zeile 25).

2.3 Objektorientiertes Datenmodell

Aufgrund der im technischen Entwicklungsbereich vorhandenen komplexen Datenstrukturen haben sich dort objektorientierte Datenmodelle als die zweckmäßigste Darstellungsform zur Datenmodellierung herausgestellt. Aus diesem Grund wird auch im WEP-Ansatz ein objektorientiertes Datenmodell verwendet. Die in einer Vererbungshierarchie angeordneten Objekte besitzen Attribute und Relationen, die Beziehungen zu anderen Objekten beschreiben. Attribute werden über ihren Namen angesprochen und sind von einem bestimmten Typ. Relationen referenzieren über ihren Namen eine Beziehung zu einem anderen Objekt der gleichen oder einer anderen Klasse.

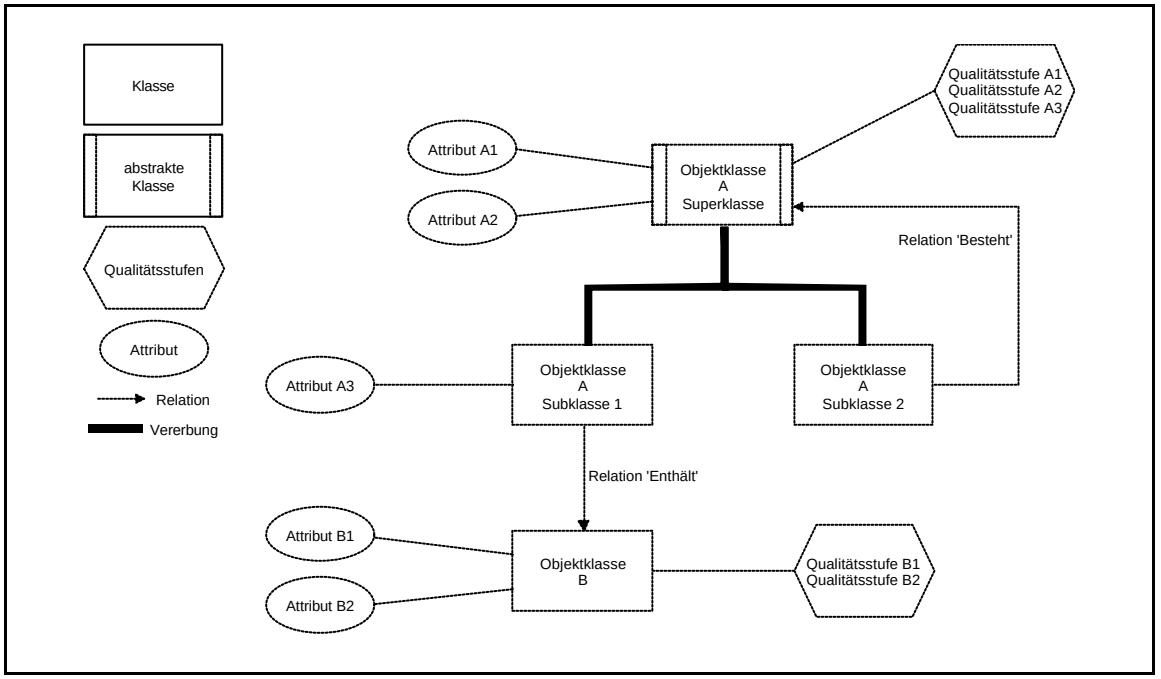


Abbildung 2.3: Abstraktes Beispiel für das objektorientierte Datenmodell von WEP

Typisch für Entwicklungsprozesse ist, daß in ihrem Verlauf die involvierten Datenobjekte immer stärker konkretisiert werden, indem weitere Attribute belegt oder zusätzliche Beziehungen aufgebaut werden. Die Ausführung von Aktivitäten ist deshalb immer nur dann sinnvoll, wenn die benötigten Eingabeobjekte einen gewissen Entwicklungsstand erreicht haben. Dieser Entwicklungsstand läßt sich am einfachsten durch die Angabe von Qualitätsstufen definieren. Deshalb unterstützt das WEP-Datenmodell zusätzlich zu Attributen und Relationen noch

Qualitätsstufen für Objekte. Abbildung 2.3 verdeutlicht die Objektstruktur des WEP-Datenmodells an einem abstrakten Beispiel.

Die Workflow-Beschreibung für die Definition einer Objektklasse ist in Abbildung 2.4 dargestellt. Zeile 1 definiert den Namen der Objektklasse, die in Zeile 3 noch zusätzlich beschrieben werden kann.

```
 1   OBJECTCLASS <ObjectClass>
 2   {
 3      DESCRIPTION <ObjectClassDescription>
 4      SUPERTYPE <ObjectClass>
 5      ATTRIBUTE <AttributeName> : <AttributeType> | <ValueList>
 6      RELATIONSHIP <RelationName> : <ObjectClass>
 7      QUALITYLEVEL <QualityLevel>
 8      {
 9         EXPRESSION <BooleanExpression>
10      }
11   }
```

Abbildung 2.4: Syntax für die Beschreibung einer Objektklasse

Charakterisiert wird ein Objektklasse in der Syntax der Workflow-Beschreibung durch die folgenden fünf Merkmale:

- **ObjectClass**: Benennung der Objektklasse (Zeile 1).
- **SuperType**: Verweist im Falle einer Vererbungshierarchie auf die abstrakte Superklasse, von der dann deren Attribute, Relationen und Qualitätsstufen geerbt werden (Zeile 4).
- **Attribute**: Definiert ein eigenes Attribut (Zeile 5) der Objektklasse durch die Angabe eines Namens (**AttributeName**) und Variablentyps (**AttributeType**). Statt des Typs kann auch eine Liste, von durch logische Oder (|) getrennten Werten angegeben werden. Diese Werteliste (**ValueList**) entspricht dann dem Wertebereich des Variablentyps. Jede Zeile dieser Art definiert ein neues Attribut.
- **Relationship**: Definiert über einen Namen (**RelationName**) eine Relation (Beziehung) zu einem anderen Objekt einer beliebigen Objektklasse (Zeile 6). Jede Relation-Zeile definiert eine neue Beziehung der Objektklasse.
- **QualityLevel**: Definiert eine Qualitätsstufe für die Objektklasse (Zeilen 7 bis 10). Sie wird mittels boolescher Ausdrücke spezifiziert (**Expression**, Zeile 9). Jede weitere Angabe dieser Blockstruktur definiert eine neue Qualitätsstufe. Wird eine Qualitätsstufe angegeben, die schon in der Superklasse definiert ist, so kann sie mittels der booleschen Ausdrücke noch genauer eingegrenzt werden.

Qualitätsstufen erlauben eine detailliertere Festlegung der vorläufigen und endgültigen Eingabe- und Ausgabeobjekten einer Aktivität, da sie mittels boolescher Ausdrücke über die Attribute des Objekts und seine Beziehungen zu anderen Objekten spezifiziert werden. Wie diese booleschen Ausdrücke generell aussehen, zeigt die Syntax in Abbildung 2.5.

```
BooleanExpression :=  AtomicExpression
                      | AtomicExpression OR AtomicExpression
                      | AtomicExpression AND AtomicExpression
                      | NOT AtomicExpression
                      | FORALL RelationName AtomicExpression
                      | EXIST RelationName AtomicExpression

AtomicExpression :=   AttributeName == Value
                      | AttributeName != Value
                      | AttributeName DEFINED
                      | AttributeName IN ValueRange
```

Abbildung 2.5: Syntax für die Spezifikation boolescher Ausdrücke

Eine Qualitätsstufe ist erreicht, wenn alle dazugehörenden Ausdrücke sich als wahr evaluieren lassen. Zwischen den Qualitätsstufen eines Objektes gibt es zwar eine Ordnung, ihre Bedingungen können sich jedoch beliebig überlappen. Folglich muß ein Objekt mit einer höheren Qualitätsstufe nicht notwendigerweise eine niedrigere Qualitätsstufe erfüllen, da die Bedingungen einer niederen Qualitätsstufe nicht notwendigerweise Teilmenge der Bedingungen einer höheren Qualitätsstufe sind.

Objektklassen, die im Workflow nicht direkt benutzt werden, die aber über Relationen innerhalb der verwendeten Objekte referenziert werden, können vereinfacht definiert werden (Abbildung 2.6). Sie werden weder von der Objektverwaltung noch von der Datenfluß-modellierung verwendet. Sie dienen nur dazu, Fehler zu vermeiden, die durch Relationen auf unbekannte Datentypen entstehen.

```
1   FOREIGNCLASS <ObjectClass>
2   {
3      DESCRIPTION <ObjectClassDescription>
4   }
```

Abbildung 2.6: Syntax für die Beschreibung einer nicht im Workflow verwendeten Objektklasse

Ein geeignetes Objektmodell muß auch die Verwaltung verschiedener Versionen eines Objekts ermöglichen. Das WEP-Workflow-Management-System unterstützt ein Versions-modell, bei dem alle Versionsableitungen immer auf genau einer Vorgängerversion basieren. Der Abstammungsgraph der Objektversionen läßt sich also als Baum darstellen. Eine speziell ausgezeichnete Vorgängerversion wird zur offiziellen Version des Objekts ernannt. Neue offizielle Objektversionen können nur von den Aktivitäten freigegeben werden, die Besitzer des Objekts sind. Eine Aktivität ist Besitzer eines Objekts, wenn sie selbst unabhängig ist, also in der Kontrollflußreihenfolge die erste Aktivität ist, die Zugriff auf das Objekt hat. Alle nachfolgenden Aktivitäten die bereits auf vorläufigen weitergegebenen Objektversionen arbeiten, sind somit von dieser Aktivität, und ihren Entscheidungen, die offizielle Objektversion betreffend, abhängig.

2.4 Kontrollfluß

Zielorientierte Aktivitäten können analog zu ‘herkömmlichen’ Aktivitäten zu einem prozeßorientierten Gesamtprozeß verknüpft werden. Aktivitäten sind dabei als Arbeitsabschnitte zu betrachten, die die logischen Schritte eines Prozesses bilden. Basis für die Ausführung eines solchen Prozesses ist der Kontrollfluß. Damit wird die für die Laufzeit wichtige Ausführungsreihenfolge der Arbeitsabschnitte (Aktivitäten) festgelegt. Der Kontrollfluß beschreibt den verhaltensbezogenen Aspekt der Workflow-Modellierung.

Das WEP-Modell unterstützt dafür mehrere Konstrukte. Die einfachste Form ist die Sequenz. Sie beschreibt eine sequentielle Ausführungsreihenfolge der zielorientierten Aktivitäten. Jede Aktivität besitzt genau eine Nachfolgeaktivität. Die Aktivierung des Nachfolgers erfolgt, sobald der Vorgänger (vorläufige) Daten freigibt, die für den Nachfolger als erforderliche Eingabedaten spezifiziert wurden. Modelliert wird eine solche Sequenz durch das Verbinden der Aktivitäten über gerichtete Kontrollflußkanten (Abbildung 2.7).

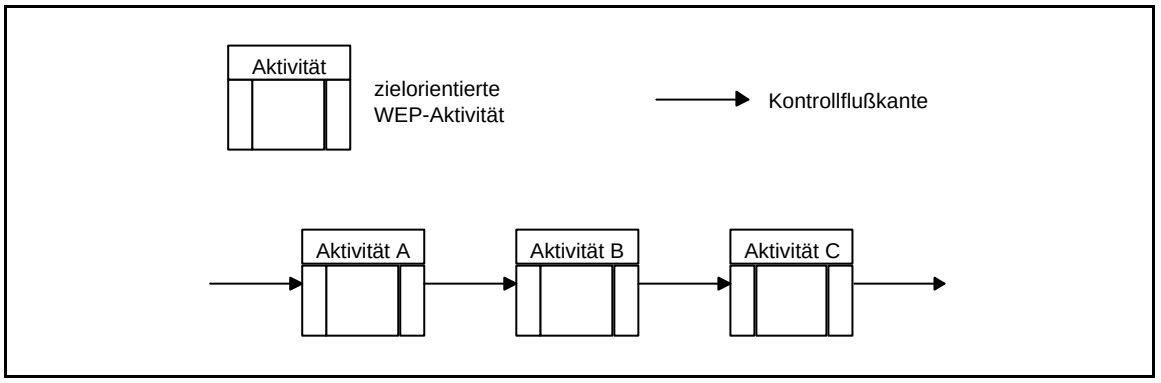


Abbildung 2.7: Kontrollfluß-Sequenz

Eine Unterstützung spezieller UND- beziehungsweise ODER-Verknüpfungen gibt es im WEP-Modell nicht. Verzweigungen kann man jedoch modellieren, indem man für eine Aktivität einfach mehrere Nachfolgeaktivitäten definiert. Dies wird grafisch durch mehrere, von einer Aktivität ausgehende Kontrollflußkanten, realisiert. Die Nachfolgeaktivitäten werden analog zur Sequenz immer dann aktiviert, wenn die Vorgängeraktivität die erforderlichen Eingabedaten zur Verfügung stellt. Zusätzlich können parallele Kontrollflüsse, die sich aus einer Verzweigung bilden, durch ein Traversierungsmerkmal eingeleitet werden. Dazu später mehr.

Um eine wiederholte Ausführung von Aktivitäten zu ermöglichen, unterstützt das WEP-Modell ein Schleifenkonstrukt. Schleifen bilden einen symmetrischen Block. Der Anfang und das Ende dieses Blocks ist durch eindeutige Start- und End-Aktivitäten gegeben. Abhängig vom Resultat (Rückgabewert) der End-Aktivität, wird die Schleife entweder verlassen oder mit einem Sprung zurück zur Start-Aktivität erneut ausgeführt (Abbildung 2.8). Sollte die Entscheidung eines erneuten Schleifendurchlaufs aufgrund von vorläufigen Daten getroffen worden sein und sind die betroffenen Aktivitäten somit noch nicht beendet, so müssen sie vor der Neubearbeitung abgebrochen werden.

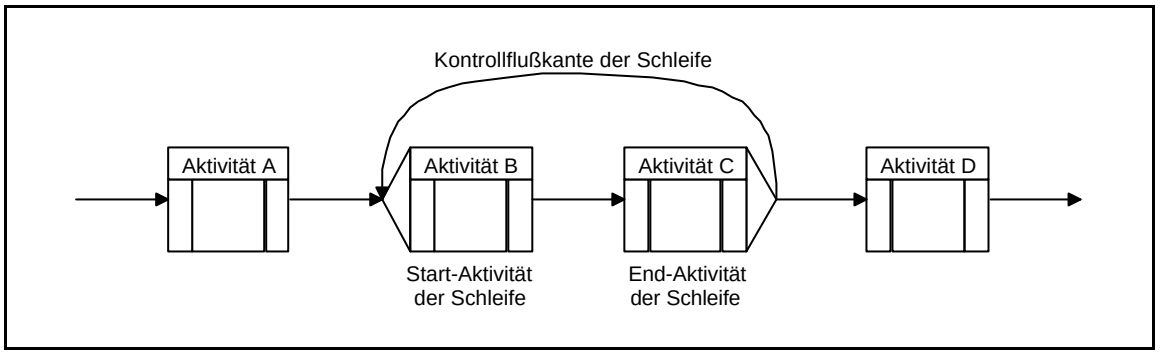


Abbildung 2.8: Kontrollfluß-Sequenz mit Schleifenkonstrukt

Abbildung 2.9 stellt die Kontrollflußmodellierung anhand der Syntax der Workflow-Beschreibungssprache dar. Beschrieben werden müssen natürlich nur noch die Kontrollflußkanten. Zeile 1 gibt der Kontrollflußkante einen Namen und in Zeile 3 kann sie noch zusätzlich beschrieben werden.

```
1   CONTROLFLOW <ControlFlowName>
2   {
3      DESCRIPTION <ControlFlowDescription>
4      FROM <ActivityName>
5      [BACK] TO <ActivityName>
6      IF RETURNCODE IS <ReturnCodeName>
7   }
```

Abbildung 2.9: Syntax für die Beschreibung einer Kontrollflußkante (einfache Version)

Charakterisiert wird eine Kontrollflußkante durch die folgenden Merkmale:

- **ControlFlowName**: Benennung der Kontrollflußkante (Zeile 1).
- **From**: Spezifiziert über einen Namen (**ActivityName**) die Vorgänger-Aktivität (Zeile 4).
- **To**: Spezifiziert über einen Namen die Nachfolgeaktivität (Zeile 5). Ist das Kontrollwort **Back** angegeben, so wird hiermit die Kontrollflußkante eines Schleifenkonstrukts definiert. Anderenfalls handelt es sich um eine Kontrollflußkante für eine Sequenz.
- **IfReturnCode**: Durch diese Angabe ist die Aktivierung der Nachfolgeaktivität von dem Resultat (**ReturnCodeName**) der Vorgänger-Aktivität abhängig. Dies entspricht einer Verzweigung.

Wie bereits in Kapitel 1 beschrieben, ist die Feinplanung eines Entwicklungsprozesses von der konkreten Ausprägung der zugrundeliegenden Objektstruktur abhängig und steht erst zur Prozeßausführung fest. Verschiedene Aufgaben in Entwicklungsprozessen werden aber nur auf einen Teil eines komplexen Objekts angewendet. Auch läßt sich die Entwicklung einer variablen Anzahl von Einzelteilen nicht durch eine vorgefertigte Kontrollflußmodellierung darstellen. Aus diesem Grund führt WEP ein zusätzliches Modellierungskonstrukt ein. Dieses Konstrukt, Traversierungsmerkmal genannt, dient zur Beschreibung von Parallelität mit einer variablen Anzahl paralleler Prozeßzweige.

Das Traversierungsmerkmal sorgt für die dynamische Aufsplittung des Kontrollflusses. Es zerlegt ein komplex strukturiertes Eingabeobjekt entlang einer festgelegten Objektbeziehung (Relation) und erzeugt entsprechend der Anzahl der entstandenen Sub-Objekte eine variable Anzahl von parallelen Sub-Prozessen. Man kann sagen, daß für jedes Sub-Objekt ein neuer Sub-Workflow gestartet wird. Wichtig dabei ist, daß die Objekttraversierung die Objektbeziehungen nicht permanent aufbricht, so daß Folgeschritte wiederum auf das gesamte komplexe Objekt zugreifen können. Die Durchführung der Traversierung kann außerdem an Bedingungen geknüpft sein, die durch einen booleschen Ausdruck spezifiziert werden. Der Ausdruck bezieht sich dabei auf die Attribute des Zielobjekttyps, auf den die Objektbeziehung zeigt.

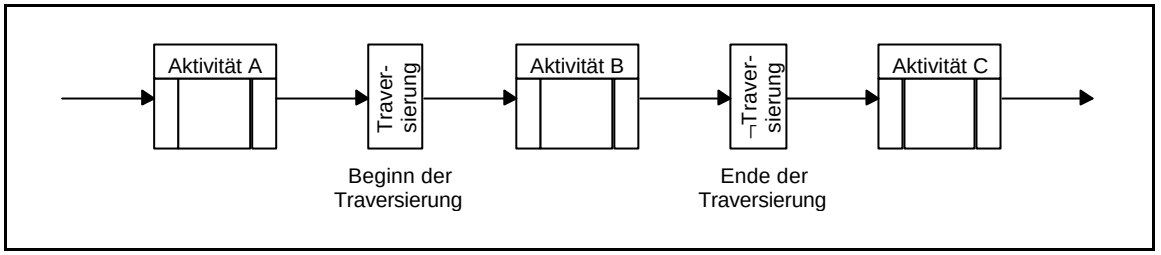


Abbildung 2.10: Kontrollfluß mit Traversierungsmerkmal

Abbildung 2.10 zeigt die Darstellung eines Kontrollflusses mit einem Traversierungs-merkmal. Aktivität B ist von der Traversierung eingeschlossen und kann deshalb in einer variablen Anzahl Instanzen gestartet werden, abhängig von der Anzahl Sub-Objekte, auf die die von der Traversierung betroffene Relation verweist. Dies sieht dann zur Laufzeit beispielsweise so aus, wie in Abbildung 2.11. Dabei sind drei Sub-Objekte von der Traversierung betroffen, wodurch drei Instanzen der Aktivität B aktiviert werden.

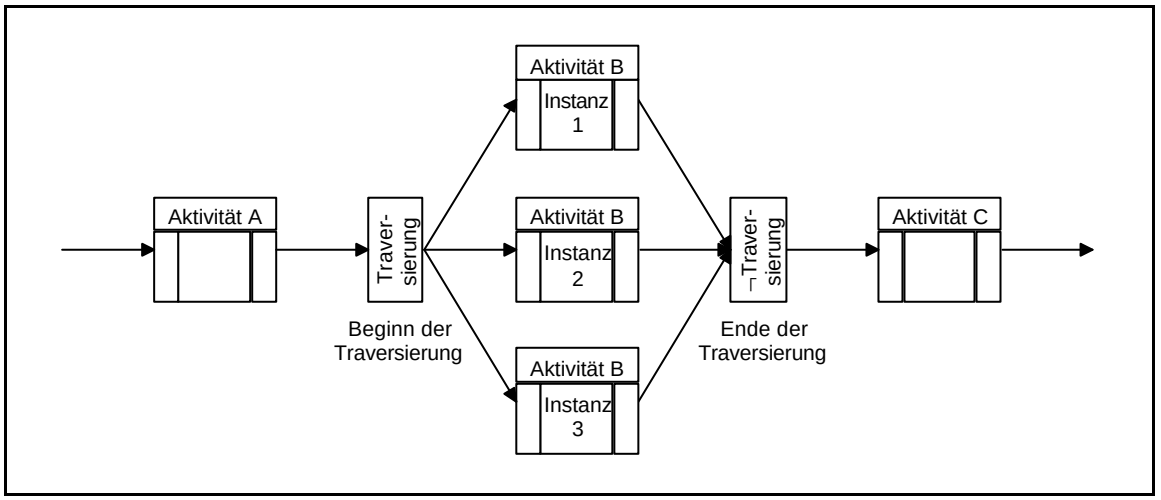


Abbildung 2.11: Kontrollfluß (zur Laufzeit) mit Aktivitäteninstanzen innerhalb einer Traversierung

```
1   TRAVERSE <TraverseName> : <ObjectClass>
2   {
3      DESCRIPTION <TraverseDescription>
4      STARTACTIVITY <ActivityName>
5      RELATION [RECURSIVE] <RelationName>
6      {
7         EXPRESSION <BooleanExpression>
8      }
9   }
```

Abbildung 2.12: Syntax für die Beschreibung eines Traversierungsmerkmals

Mit der Workflow-Beschreibungssprache erfolgt die Definition eines Traversierungsmerkmals wie in Abbildung 2.12. Die charakteristischen Merkmale sind:

- **TraverseName**: Benennung des Traversierungsmerkmals (Zeile 1). Über diesen Namen wird das Traversierungsmerkmal von den Kontrollfluß- und Datenflußstrukturen referenziert.
- **ObjectClass**: Objektklasse, auf der die Traversierung ausgeführt wird (Zeile 1).
- **StartActivity**: Gibt mittels eines Namens (**ActivityName**) die Aktivität an, mit der der Sub-Workflow der Traversierung gestartet wird (Zeile 4). Ausgehend von dieser Aktivität entwickelt sich der parallele Kontrollfluß.
- **Relation**: Spezifiziert über den Relationsnamen (**RelationName**) die Beziehung entlang der die Objektklasse aufgebrochen wird (Zeilen 5 bis 8). Durch Angabe des Kontrollwortes **Recursive** (Zeile 5) wird die Relation im Falle von rekursiven Datenstrukturen mehrfach durchlaufen. Innerhalb einer Traversierung können auch mehrere Relationen angegeben werden. Siehe dazu auch denn folgenden Abschnitt.
- **Expression**: Durch einen booleschen Ausdruck kann die Traversierung an eine Bedingung geknüpft werden, die sich auf den Zielobjekttyp (ObjectClass) bezieht.

Eine Traversierung ist nicht auf eine einzelne Objektbeziehung begrenzt. Sie kann sich auch über mehrere Relationen erstrecken. Dabei muß jedoch festgelegt werden, in welcher Reihenfolge die Relationen zum Zuge kommen. Das heißt, die Relationen werden für jeden entstehende parallelen Kontrollfluß in einer festgelegten Reihenfolge durchlaufen. In der Workflow-Beschreibung ist die Aufschreibereihenfolge von Belang. Zum Traversieren rekursiver Datenstrukturen kann ein Relationstyp auch mehrfach durchlaufen werden. Das Traversieren endet aber auf jeden Fall immer dann, wenn man an einem Objekt angekommen ist, das keine entsprechende Relation mehr vorweisen kann.

Für die Einführung des Traversierungsmerkmals muß die Kontrollfluß-Syntax der Workflow-Beschreibungssprache noch etwas erweitert werden (Abbildung 2.13).

```
1   CONTROLFLOW <ControlFlowName>
2   {
3      DESCRIPTION <ControlFlowDescription>
4      FROM <ActivityName> | <TraverseName>
5      [BACK] TO <ActivityName> | <TraverseName> | TRAVERSEEND
6      IF RETURNCODE IS <ReturnCodeName>
7   }
```

Abbildung 2.13: Syntax für die Beschreibung einer Kontrollflußkante (vollständige Version)

Wie in den Zeilen 4 und 5 dargestellt, sind jetzt auch Traversierungsmerkmale als ‘Quelle’ und als ‘Ziel’ innerhalb des Kontrollflusses zulässig. Desweiteren muß das Ende eines Sub-Workflows innerhalb einer Traversierung spezifiziert werden. Dies geschieht durch Angabe des speziellen Ziels TRAVERSEEND (Zeile 5).

2.5 Datenfluß

Neben der Kontrollflußmodellierung erfolgt beim WEP-Modell auch eine Datenflußmodellierung. Dabei werden die Ein- und Ausgabeobjekte der zielorientierten Aktivitäten mit Referenzen auf globale Datenobjekte versehen.

Das WEP-Modell unterstützt drei Konstrukte für die Datenflußmodellierung. Dies ist die Definition globaler Datenobjekte, Datenflußbeziehungen von den Ausgabeobjekten der Aktivitäten zu den globalen Datenobjekten und Datenflußbeziehungen von den globalen Datenobjekten zu den Eingabeobjekten der Aktivitäten. Abbildung 2.14 zeigt alle drei Datenflußkonstrukte anhand einer Kontrollfluß-Sequenz.

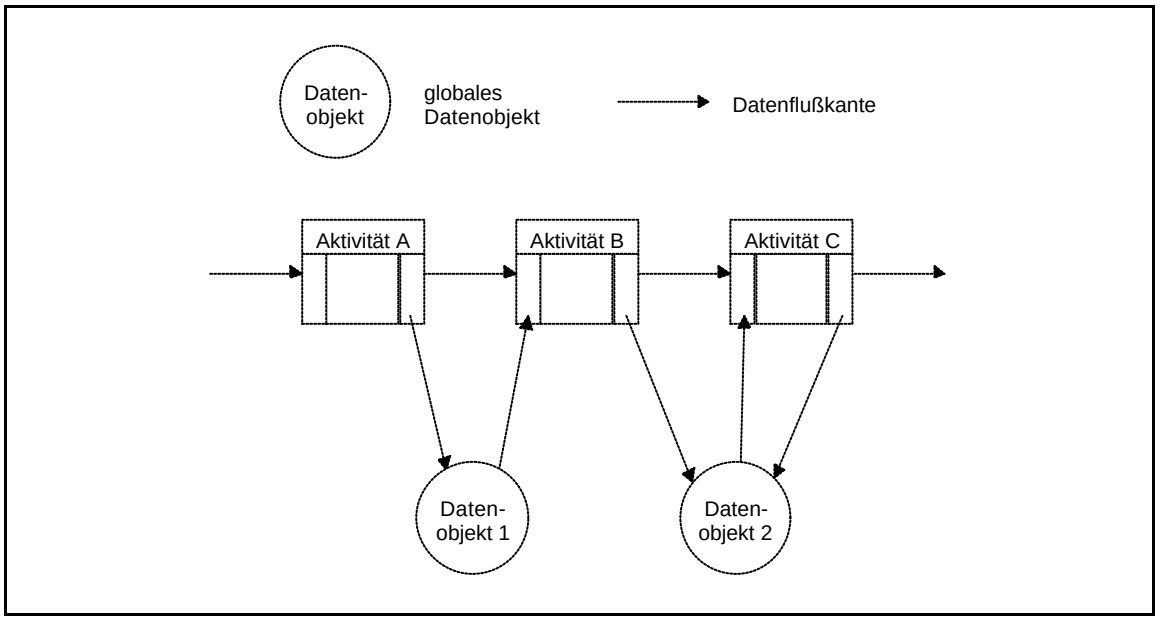


Abbildung 2.14: Kontrollfluß einer Sequenz mit zugehörigem Datenfluß

Dargestellt sind zwei globale Datenobjekte. Aktivität A schreibt die Daten ihres Ausgabeobjekts in das globale Datenobjekt 1. Diese Daten werden danach wieder als Eingabeobjekt von Aktivität B gelesen. Aktivität B schreibt ihr Ausgabeobjekt in das globale Datenobjekt 2. Aktivität C liest diese Daten als ihr Eingabeobjekt und beschreibt mit ihrem Ausgabeobjekt erneut das globale Datenobjekt 2.

Beschreibt man den Vorgang auf Basis der Eingaben und Ausgaben der Aktivitäten, so bekommt Aktivität B die Ausgaben von Aktivität A als Eingaben. Die Ausgaben von Aktivität B dienen dagegen Aktivität C als Eingaben. Das Ergebnis der gesamten Sequenz liegt am Schluß im globalen Datenobjekt 2. Die Datenflußkanten in der grafischen Darstellung symbolisieren diesen Ablauf.

Globale Datenobjekte dienen eigentlich lediglich der Bindung eines im WEP-Workflow spezifizierten komplexen Datenobjekts mit einem Namen. Sie spezifizieren also die Variablen, beziehungsweise den Speicherplatz, der dem Workflow zugewiesen wurde. Mittels eines Namens oder einer Referenz kann direkt über das, unter dem WEP-Workflow-Management-System liegende Datenhaltungssystem, auf die Variablen zugegriffen werden. Abbildung 2.15 stellt die Syntax für die Definition eines globalen Datenobjekts dar.

```
1   GLOBALOBJECT <GlobalObjectName> : <ObjectClass>
2   {
3      DESCRIPTION <GlobalObjectDescription>
4      DBNAME <ObjectReference>
5      TRAVERSED OF <GlobalObjectName>.<TraverseName>
6   }
```

Abbildung 2.15: Syntax für die Beschreibung eines globalen Datenobjekts

Charakteristisch für ein globales Datenobjekt sind die folgenden Merkmale:

- **GlobalObjectName**: Benennung des globalen Datenobjekts (Zeile 1). Über diesen Namen wird es in Datenflußbeziehungen referenziert.
- **ObjectClass**: Objektklasse des globalen Datenobjekts (Zeile 1). Wird im Datenmodell definiert.
- **DbName**: Name oder Referenz (**ObjectReference**) des globalen Datenobjekts im Datenhaltungssystem (Zeile 4).
- **TraversedOf**: Ist das Eingabe- beziehungsweise Ausgabeobjekt der Aktivität, das mit diesem globalen Datenobjekt verbunden wird, durch Traversierung entstanden, so muß dies zur Verdeutlichung der Herkunft spezifiziert werden (Zeile 5). Dies geschieht durch Angabe des Traversierungsmerkmals (**TraverseName**) und dem Namen eines globalen Datenobjekts (**GlobalObjectName**), von dessen Relation die Traversierung ausgeht. Siehe dazu auch Abbildung 2.17 mit Erklärung.

Datenflußbeziehungen (Datenflußkanten) beschreiben die Abhängigkeiten zwischen den Ein- und Ausgabeobjekten einer Aktivität mit den globalen Datenobjekten des Workflows. Sie definieren also den Speicherort, an dem die Ausgabedaten einer Aktivität während dem Übergang zu einer neuen Aktivität, zwischengespeichert werden. Die Eingabedaten der Folgeaktivität werden dann wieder aus diesem Speicherort gelesen. Abbildung 2.16 beschreibt die Syntax für eine Datenflußbeziehung.

```
1   DATAFLOW <DataFlowName>
2   {
3      DESCRIPTION <DataFlowDescription>
4      TYPE IS INPUT | OUTPUT
5      ACTIVITYOBJECT <ActivityName>.<ObjectName>
6      GLOBALOBJECT <GlobalObjectName>
7   }
```

Abbildung 2.16: Syntax für die Beschreibung einer Datenflußbeziehung

Für eine Datenflußbeziehung sind die folgenden Merkmale charakteristisch:

- **DataFlowName**: Benennung der Datenflußkante (Zeile 1).
- **Type**: Definiert die Art der Datenflußkante (Zeile 4). Vom globalen Datenobjekt zum Eingabeobjekt einer Aktivität (INPUT) oder vom Ausgabeobjekt einer Aktivität zum globalen Datenobjekt (OUTPUT).
- **ActivityObject**: Spezifiziert mittels Aktivitätenname (**ActivityName**) und Objektname (**ObjectName**) das betroffene Aktivitäten-Objekt (Zeile 5). Durch das Merkmal **Type** wird festgelegt, ob es sich dabei um ein Eingabe- oder Ausgabeobjekt handelt.
- **GlobalObject**: Spezifiziert über den Namen das betroffene globale Datenobjekt (Zeile 6).

Im Normalfall werden die Ausgabeobjekte einer Aktivität immer als ganzes auf die Eingabeobjekte von Folgeaktivitäten abgebildet (siehe Abbildung 2.14). Durch die Erweiterung des Datenmodells um Qualitätsstufen, müssen nun jedoch die ‘Qualitätsansprüche’ der zielorientierten Aktivitäten mit berücksichtigt werden. Zielorientierte Aktivitäten werden also nicht immer sofort aktiviert, sobald die Datenflußkanten für ihre Eingabeobjekte anliegen. Es muß zusätzlich überprüft werden, ob die bereitstehende Datenqualität des globalen Datenobjekts auch mit der von der Aktivität geforderten Datenqualität des Eingabeobjekts übereinstimmt. Eine weitere Ausnahme entsteht beim Aufbrechen einer Objektbeziehung über ein Traversierungsmerkmal.

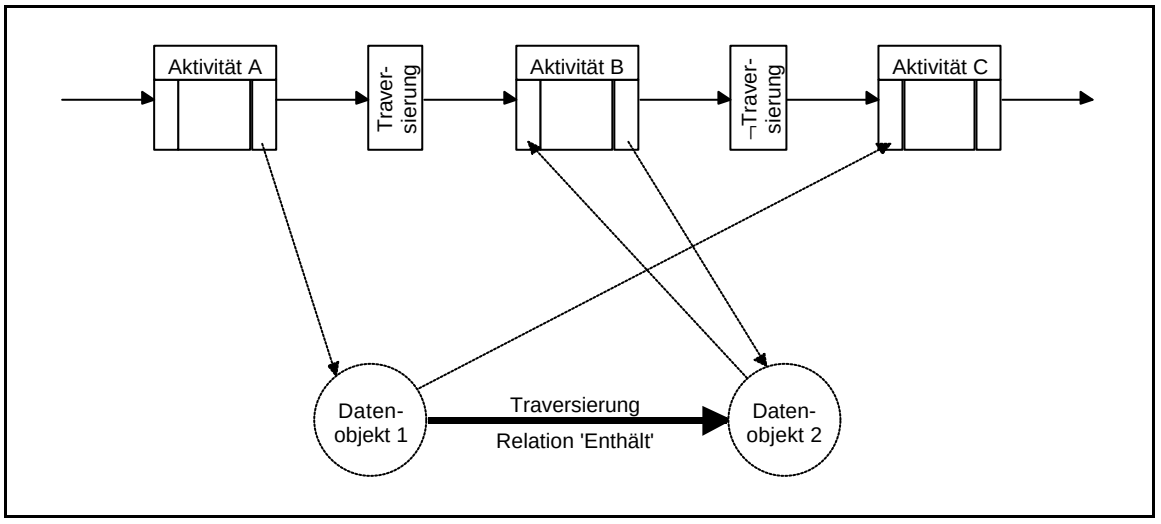


Abbildung 2.17: Kontrollfluß mit Traversierungsmerkmal und zugehörigem Datenfluß

Abbildung 2.17 zeigt einen Kontrollfluß mit Traversierungsmerkmal und den zugehörigen Datenfluß. Aktivität B ist von der Traversierung eingeschlossen und kann deshalb in einer

variablen Anzahl Instanzen gestartet werden, abhängig von der Anzahl Sub-Objekte, auf die die Relation ‘Enthält’ des globalen Datenobjekts 1 verweist. Vom globalen Datenobjekt 2, das das Sub-Objekt darstellt, entstehen zur Laufzeit ebensoviele Instanzen wie von Aktivität B. Um diese Beziehung zwischen den zwei globalen Datenobjekten festzuhalten, muß in der Syntax-Beschreibung des Sub-Objekts das Merkmal *TraversedOf* passend angegeben werden (siehe Abbildung 2.15). Der Datenfluß macht deutlich, daß nach dem Ende der Traversierung Aktivität C wieder auf das gesamte Datenobjekt zugreifen kann.

Anhand der beschriebenen Konstrukte des Kontrollflusses und Datenflusses kann das WEP-Workflow-Management-System bereits zur Modellierungszeit die Korrektheit des Workflows überprüfen. Alle definierten Aktivitäten müssen im Kontrollfluß eingebunden sein und alle globalen Datenobjekte, sowie die Eingabe- und Ausgabeobjekte der Aktivitäten müssen über Datenflußbeziehungen verbunden sein. Daraus sollte sich ergeben, daß die Eingabeobjekte nachfolgender Aktivitäten bereits bei früheren Aktivitäten als Ausgabeobjekte definiert sind. Dies trifft genauso für die Datenqualitäten zu. Datenqualitäten, die von Nachfolge-Aktivitäten benötigt werden, müssen auch von deren Vorgängern bereitgestellt werden können.

2.6 Zeit-Management

Ein Zeitmanagement ist eine wichtige Anforderung für die Unterstützung von Entwicklungsprozessen, da diese zu bestimmten Zeitpunkten vordefinierte Zustände erreichen müssen. Diese Zeitpunkte werden im WEP-Modell als Meilensteine innerhalb zielorientierter Aktivitäten definiert (siehe Kapitel 2.2). Diese aktivitätenlokalen Zeitpunkte sind innerhalb einer zielorientierten Aktivität immer relativ zum Startzeitpunkt der Aktivität zu sehen (Abbildung 2.18).

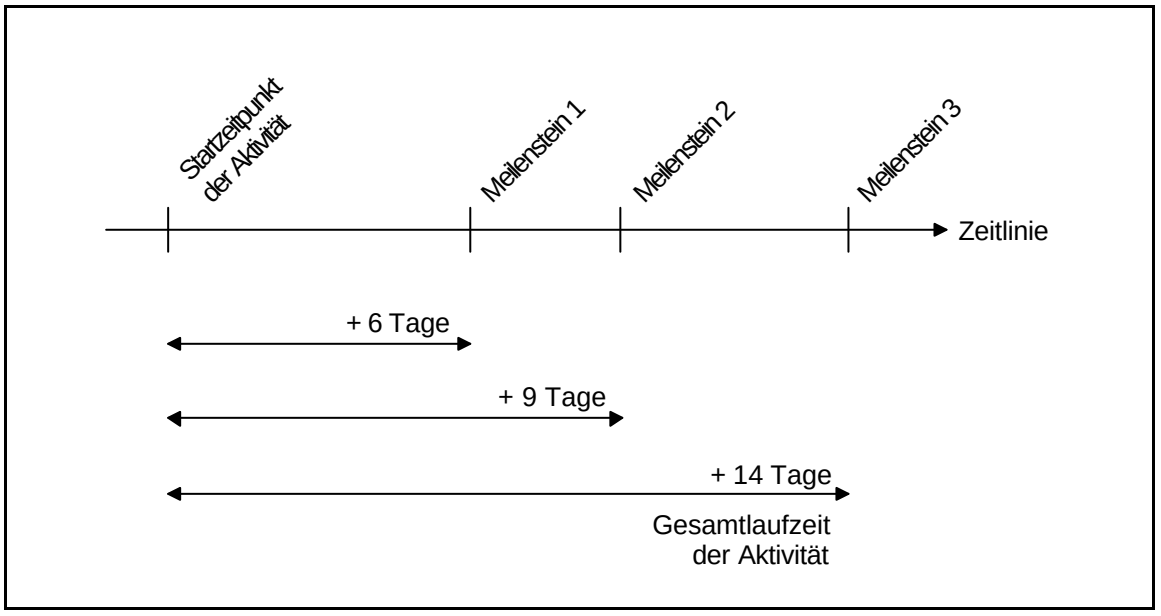


Abbildung 2.18: Relative Zeitangaben auf Aktivitäten-Ebene

Alle Zeitpunkte zur Modellierungszeit sind relativ. Beim Start des Workflows müssen diese logischen Zeitpunkte durch entsprechende Umrechnung in reale Zeitpunkte umgewandelt

werden. Dies geschieht in der Regel durch das Ermitteln der aktuellen Zeit. Nun können für alle Aktivitäten entlang des Kontrollflusses die realen Zeitpunkte ermittelt werden. Die Gesamtlaufzeit (Bearbeitungsdauer) einer Aktivität ergibt sich dabei aus der höchsten Zeitangabe eines Meilensteins. Rechnet man alle Laufzeiten der Aktivitäten im Kontrollfluß zusammen, so kann man die Laufzeit des Workflows bestimmen. Wird beim Start des Workflows der Zeitpunkt übergeben, an dem das Ergebnis des Workflows bereitstehen muß, so kann jetzt errechnet werden, ob der Workflow innerhalb der Zeitspanne erfolgreich ausgeführt werden kann. Überschreitet ein Bearbeiter den festgelegten Zeitpunkt eines Meilensteins, so muß er vom System davon in Kenntnis gesetzt werden. Davon abhängige Zeitpunkte in Folge-Aktivitäten müssen dann eventuell korrigiert werden.

Problematisch für das Zeitmanagement im Falle von Schleifenkonstrukten. Durch Schleifen entstehen theoretisch unendlich lange Bearbeitungszeiten. Hier unterstützt das WEP-Workflow-Management-System die Angabe von Verzweigungswahrscheinlichkeiten.

Eine detaillierte Betrachtung des Zeitmanagements war nicht Ziel dieser Diplomarbeit und wurde deshalb nicht durchgeführt. Es ist aber geplant, eine weitere Diplomarbeit zu stellen, die auch diese Aspekte betrachtet.

2.7 Rollenzuordnung

Im WEP-Workflow-Management-System geschieht die Zuordnung eines Bearbeiters zu einer Aktivität nicht direkt über einen Namen, sondern über einen Rollennamen. Dies bedeutet, daß sich ein Bearbeiter unter einem Rollennamen, zum Beispiel Designer oder Konstrukteur, beim System anmelden muß. Von diesem bekommt er dann solche Aktivitäten zugewiesen, die für diesen Rollennamen freigegeben sind. Welche Rolle eine Aktivität bearbeiten kann wird mit der Aktivitätenbeschreibung festgelegt (siehe Kapitel 2.2 Zielorientierte Aktivitäten).

Die für einen Workflow erlaubten Rollennamen werden wie in Abbildung 2.19 angegeben. Zeile 1 definiert den Rollennamen. In Zeile 3 kann das Aufgabenfeld der Rolle noch detaillierter beschrieben werden.

```
1   USERROLE <UserRoleName>
2   {
3      DESCRIPTION <UserRoleDescription>
4   }
```

Abbildung 2.19: Syntax für die Beschreibung eines Rollennamens

Für den in dieser Diplomarbeit entwickelten Prototyp eines WEP-Workflow-Management-Systems ist dieses einfache Rollenkonzept ausreichend. Für ein im Routinebetrieb eingesetztes System ist selbstverständlich ein viel komplexeres Organisationsmodell mit Organisationsstrukturen, Stellvertreterregelungen, Kompetenzen, und dergleichen notwendig (vgl. dazu auch [Ditt95], [Kubi98]).

2.8 Benutzerinteraktionen

Die zielorientierten Aktivitäten des WEP-Modells ermöglichen die Spezifikation vorläufiger Ergebnisdaten und bilden damit die Grundlage für zusätzliche Interaktionsfunktionalität. Diese weitergehenden Benutzerinteraktionen ermöglichen die Weitergabe vorzeitiger Datenobjekte vor Beendigung der Aktivität, die Rücknahme vorzeitig weitergegebener Datenobjekte, die Anforderung neuer Objektversionen und eine Konsolidierungsrunde (Abstimmungsrunde), in der die betroffenen Bearbeiter über eine neue Objektversion entscheiden können.

Alle Arten der erweiterten Funktionalität tragen dazu bei, daß in einem WEP-Workflow-Management-System sequentielle Bearbeitungsschritte überlappend ausgeführt werden können. Damit lassen sich die Prozeßdurchlaufzeiten verkürzen. Während dieser Simultaneous-Engineering-Phasen muß das WEP-System diesen Ablauf überwachen und steuern. Dabei muß darauf geachtet werden, welche Aktivitäten welche vorläufigen Daten erhalten haben (abhängige Aktivitäten). Die Bearbeiter dieser abhängigen Aktivitäten müssen über neue Objektversionen und damit auch über neu ausführbare Aktivitäten unterrichtet werden. Weiterhin muß das Zusammenführen autonom entstandener Objektversionen und Abstimmungsrunden über die richtige Objektversion verwaltet werden.

2.8.1 Standardinteraktionen

Die zusätzliche Funktionalität wird natürlich ebenso wie die Benutzeraktionen, die der Funktionalität von Standard-Workflow-Management-Systemen entsprechen, dem Bearbeiter zur Verfügung gestellt. Zunächst wird eine typische Interaktionsreihenfolge für einen Standard-Workflow, ohne die zusätzliche Funktionalität von WEP, beschrieben. Das Vorgehen wird dabei am einfachsten anhand von Operationen und einer Beispiel-Oberfläche beschrieben (siehe Abbildung 2.20).

In der Abbildung ist eine Workliste für einen Bearbeiter zu sehen. In ihr werden immer die Aktivitäten aufgelistet, die vom Bearbeiter ausgeführt werden können. Eine Aktivität wird mit der Oberfläche eines möglichen Klienten gezeigt (U-Maske). Dabei werden im linken Feld (Input-Area) die Eingabeobjekte und im rechten Feld (Output-Area) die Ausgabeobjekte eingetragen. In der Mitte sind die möglichen Werkzeuge (Programmschritte) und die gerade bearbeiteten Objekte (Work-Area) dargestellt. Die Reihenfolge der Operationen gibt den normalen Ablauf wieder.

Zuerst muß der Workflow initialisiert werden (*InitWorkflow*). Dabei wird dem System die Workflow-Beschreibung und eine Zeitangabe übergeben, die das Enddatum des Entwicklungsprozesses definiert. Von Anfang an ausführbare Aktivitäten werden daraufhin in die Worklisten der Bearbeiter, die dafür in Frage kommen, eingetragen.

Ein Bearbeiter kann eine solche Aktivität mit der Operation *StartActivity* aufrufen. Daraufhin hat er Zugriff auf die gezeigte Aktivitätenoberfläche. Bei den Eingabeobjekten ist dem Bearbeiter ersichtlich, mit welcher Datenqualität sie belegt sind und ob es sich um vorzeitig weitergegebene Objekte handelt. Die Ausgabeobjekte werden durch Meilensteine beschrieben, also wann sie in welcher Qualität zur Verfügung gestellt werden müssen.

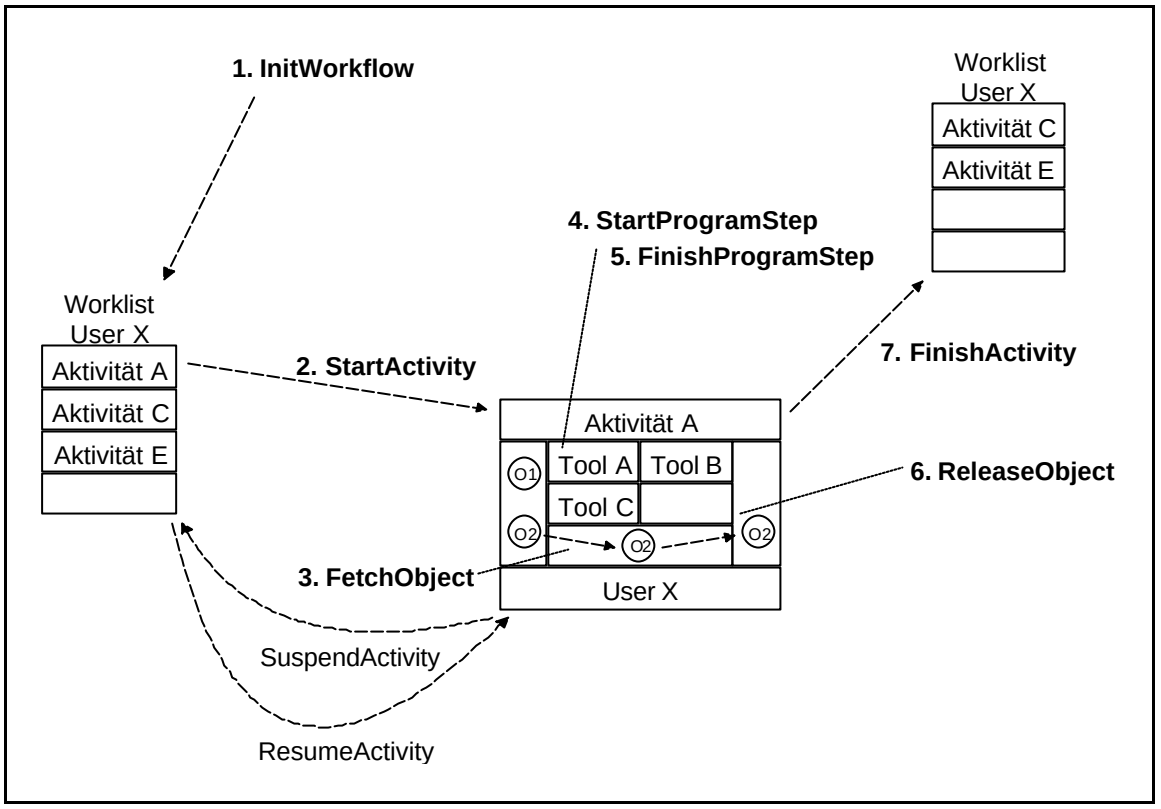


Abbildung 2.20: Standardinteraktionsabfolge bei WEP

Die Eingabeobjekte stehen dem Bearbeiter grundsätzlich nur zum Lesen zur Verfügung. Zur Manipulation eines Objektes muß er erst mit der Operation *FetchObject* eine für ihn änderbare Objektversion erzeugen. Das Objekt wird dann in die Work-Area übernommen. Handelt es sich bei der Aktivität um eine unabhängige Aktivität, die nicht von den Objektversionen anderer Aktivitäten abhängt, so hat der Bearbeiter nun auch das Recht, neue offizielle Objektversionen festzulegen.

Da zielorientierte Aktivitäten im Vergleich zu den Elementaraktivitäten 'herkömmlicher' Workflow-Modelle weit komplexere Aufgaben repräsentieren, kann ihnen nicht automatisch die Ausführung eines Werkzeugs zugeordnet werden. Zielorientierte Aktivitäten enthalten mehrere Werkzeuge, die vom Bearbeiter in freier Reihenfolge und Anordnung ausgeführt werden können. Festgelegt sind nur die erforderlichen Eingabeobjekte für ein Werkzeug. Der Aufruf eines Werkzeuges innerhalb einer zielorientierten Aktivität muß also durch spezielle Operationen gesteuert werden. So führt der Aufruf von *StartProgramStep* ein Werkzeug auf einem oder mehreren Objekten in der Work-Area aus. Mit *FinishProgramStep* kann die Ausführung wieder beendet werden.

Sind die Arbeiten am Objekt beendet, so kann es mit *ReleaseObject* wieder freigegeben werden. Dabei wird es von der Work-Area in die Output-Area übernommen. Systemseitig muß natürlich sichergestellt sein, daß das 'fertige' Objekt den Anforderungen, das heißt der erforderlichen Ausgabequalität entspricht. Die weitergehenden Möglichkeiten von ReleaseObject werden im folgenden Kapitel 2.8.2 aufgezeigt.

Ist ein Bearbeiter schließlich der Meinung, seine Arbeit wäre beendet, so kann er die Aktivität mit der Operation *FinishActivity* beenden. Dabei werden alle belegten Objekte freigegeben. Dem Bearbeiter wird eine Liste möglicher Aktivitätsresultate (Returncodes) angeboten, aus der er das seiner Meinung nach passende auswählen muß. Die angebotenen Returncodes sind von den Qualitätsstufen der in der Output-Area enthaltenen Ausgabeobjekte abhängig.

Zusätzlich zu diesen Standardinteraktionen sind in Abbildung 2.20 noch die Operationen *SuspendActivity* und *ResumeActivity* dargestellt. Sie erlauben es einem Bearbeiter, längerdauernde Aktivitäten zeitweise zu unterbrechen und später wieder fortzusetzen.

2.8.2 Vorzeitige Datenweitergabe

Zusätzliche Operationen und eine Erweiterung der Funktionalität der Operation *ReleaseObject* erlauben in WEP die vorzeitige Datenweitergabe und die Arbeit auf vorzeitig weitergegebenen Objekten in den dann abhängigen Aktivitäten. Diese Vorgänge werden in Abbildung 2.21 anhand der in der vorigen Abbildung bereits erklärten Symbolik dargestellt.

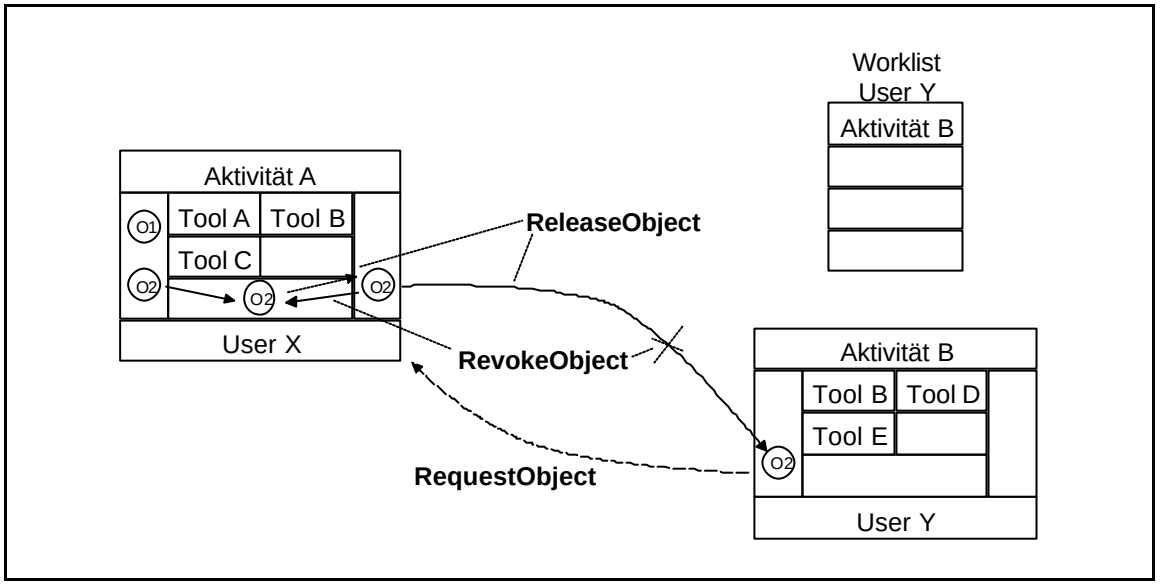


Abbildung 2.21: Interaktionsformen bei vorzeitiger Datenweitergabe

Erfüllen mit *ReleaseObject* von der Work-Area in die Output-Area übernommene Objekte die vormodellierten Datenqualitäten, so können sie für Folgeaktivitäten (abhängige Aktivitäten) als vorläufige Eingabeobjekte weitergegeben werden, so daß Simultaneous-Engineering-Phasen möglich werden. Mehrere Aktivitäten, beziehungsweise Bearbeiter, arbeiten jetzt auf ein und demselben Objekt. In der Abbildung ist zu sehen, wie die abhängige Aktivität B nach dem Aufruf der Operation *ReleaseObject* in die Workliste eines neuen Bearbeiters eingetragen wird. Dieser kann die Aktivität starten und auf dem vorzeitigen Datenobjekt mit seiner Arbeit beginnen.

Weitergegebene Objekte können mit der Operation *RevokeObject* durch den Bearbeiter, der das Objekt ursprünglich freigegeben hat, wieder zurückgenommen werden. Das Objekt wird dadurch von der Bearbeitung durch Folgeaktivitäten gesperrt. Eine neue Objektversion wird dabei nicht zur Verfügung gestellt. Diese Aktion ist immer dann zweckmäßig, wenn eine Objektüberarbeitung zeitaufwendig, eine Fortführung mit einer alten Version aber nicht sinnvoll ist.

Die zweite zusätzliche Operation ist *RequestObject.* Sie erlaubt es abhängigen Aktivitäten optionale Eingabeobjekte oder bereits erhaltene Objekte in aktuellerer Version anzufordern. Damit können Bearbeiter von Folgeaktivitäten signalisieren, daß sie auf bestimmte Objekte

warten. Wie eine abhängige Aktivität auf neue Objektversionen reagiert, kann über das Merkmal *ConsistencyPolicy* von Eingabeobjekten definiert werden. Die verschiedenen Vorgehen werden im folgenden Kapitel 2.8.3 beschrieben.

2.8.3 Integrationsphasen

Zur Verkürzung der Prozeßdurchlaufzeiten mit Simultaneous-Engineering-Phasen müssen natürlich geeignete Mechanismen zur Datenkonsistenzsicherung bereitgestellt werden. In WEP können diesbezüglich Inkonsistenzen bei Folgeaktivitäten auftreten, die auf Basis vorläufiger Objektversionen gearbeitet und eventuell bereits selber neue Objektversionen erstellt haben. Ändern sich bei solchen Aktivitäten die Eingabeobjekte, so müssen die dadurch entstandenen Dateninkonsistenzen in speziellen Integrationsphasen aufgelöst werden. Das WEP-Modell sieht dafür zwei Varianten vor. Definiert wird dies über das Merkmal *ConsistencyPolicy* in der Workflow-Beschreibung der Eingabeobjekte einer zielorientierten Aktivität (Kapitel 2.2).

Der erste Lösungsansatz ist das **Undo-Redo**-Verfahren. Dabei wird mit der am weitesten fortgeschrittenen Aktivität begonnen, die mit dem betroffenen Objekt arbeitet, und bis zum direkten Nachfolger rückwärts gegangen. Es beruht auf der Grundidee der Kompensation (vgl. [GaSa87], [GGK+91]). Dabei wird einfach die von der abhängigen Aktivität erzeugte Objektversion als ungültig markiert (Undo) und die Aktivität danach erneut zur Bearbeitung angeboten (Redo). Eine echte Kompensation ist durch die Komplexität einer zielorientierten Aktivität nur in Ausnahmefällen möglich.

Undo-Redo-Verfahren muß eine bereits ausgeführte Arbeit im Normalfall erneut bearbeitet werden, was bei langdauernden Aktivitäten natürlich sehr aufwendig ist. Beim **Merge**-Integrationsverfahren wird deshalb versucht, inkonsistente Objektversionen abzugleichen. Durch Vergleich der Objektversionen müssen die Unterschiede ermittelt und die Versionen dann geeignet aneinander angeglichen werden. Sind dem System die Differenzen von Objektversionen bekannt, kann dies automatisch geschehen. Im Normalfall ist man aber auf die Unterstützung spezieller Werkzeuge (semi-automatisch) oder manuelle Differenzierung angewiesen. Moderne CAD-Tools, wie beispielsweise CATIA, unterstützen einen semi-automatischen Abgleich, indem sie verschiedene Objektversionen grafisch direkt gegenüberstellen.

2.8.4 Konsolidierungsphasen

Bei der Beschreibung eines Meilensteins von einem Ausgabeobjekt einer zielorientierten Aktivität (Kapitel 2.2), legt das Merkmal *Concurrent Mode*, die Strategie fest, wie Änderungen an vorzeitig weitergegebenen Objekten behandelt werden.

Im Modus **Autonomous** kann die weitergebende Aktivität eigenmächtig die offizielle Version des weitergegebenen Objekts aktualisieren. Dabei werden die Folgeaktivitäten erst nach der Änderung von der neuen Objektversion informiert.

Manchmal ist es jedoch sinnvoller, folgenden Aktivitäten ein Mitbestimmungsrecht einzuräumen. Dies wird vom Modus **Consolidation** unterstützt. Dieses Verfahren kommt einer

Abstimmungsrunde gleich, wobei ein Bearbeiter einen Vorschlag einbringt und alle anderen diesem zustimmen oder ihn ablehnen können (Konsolidierungsphase, Konsolidierungsrunde). Abbildung 2.22 stellt den Ablauf einer solchen Konsolidierungsrunde in WEP bildlich dar.

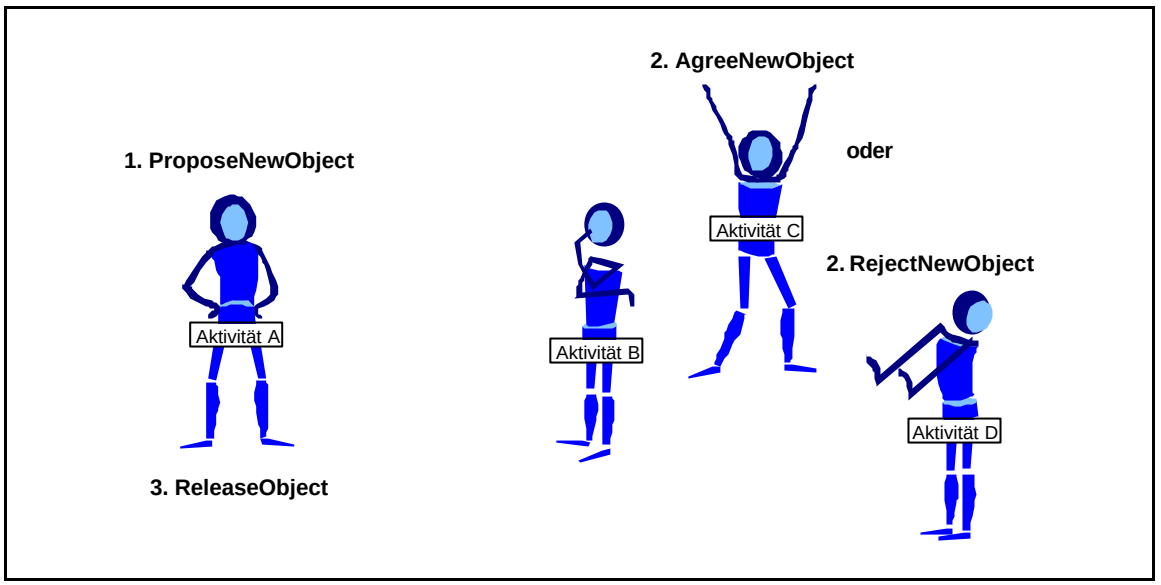


Abbildung 2.22: Ablauf einer Konsolidierungsphase

Durch den Aufruf der Operation *ProposeNewObject* wird eine neue Objektversion vorgeschlagen. Dies kann sowohl von der weitergebenden Aktivität, also auch von einer folgenden Aktivität geschehen. Dabei ist es egal ob es sich bereits um ein fertiges Objekt oder nur um die textuelle Beschreibung einer Änderung handelt. Alle ebenfalls auf dem betroffenen Objekt arbeitenden Aktivitäten, beziehungsweise Bearbeiter, müssen dieser neuen Objektversion nun mittels *AgreeNewObject* zustimmen oder sie mit *RejectNewObject* ablehnen. Ob der Vorschlag zu einer neuen offiziellen Objektversion führt, liegt aber dennoch in der alleinigen Entscheidung der weitergebenden Aktivität. Eine neue Version wird dann durch die Operation *ReleaseObject* bestätigt. Eine Konsolidierungsphase kann durch die Angabe einer Zeitschranke begrenzt werden. Damit wird sichergestellt, daß das Abstimmungsergebnis nicht von einigen unentschlossenen Aktivitäten (im Bild Aktivität B) beliebig lang verzögert wird.

WEP stellt für die Abstimmungsrunde nur Basismechanismen zur Verfügung. Somit ist nicht festgelegt, mit welchen Mitteln die Abstimmung durchgeführt wird. Dies kann in Form eines einfachen Email-Austauschs als auch durch den Einsatz eines komplexen Konferenzsystems geschehen.

2.9 Beispiel in WEP

Nachdem wir die Konzepte des WEP-Modells betrachtet haben, wollen wir versuchen ein realistisches Beispiel für einen Produktentwicklungsprozeß damit darzustellen. Es wird im folgenden natürlich vereinfacht skizziert. Die Beispiele, die in der Syntax der Workflow-Beschreibungssprache dargestellt werden, orientieren sich diesbezüglich an den Definitionen der vorigen Kapitel.

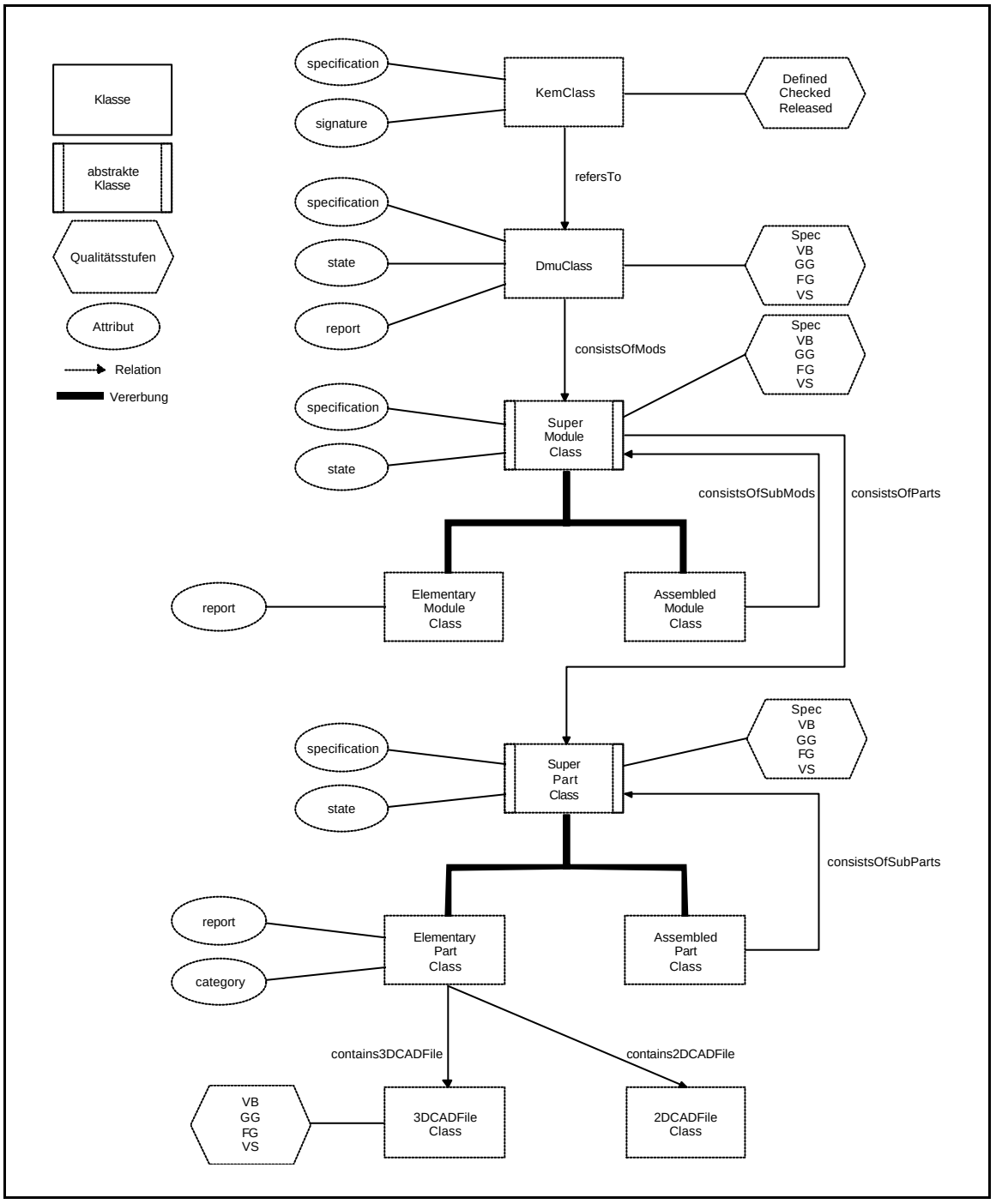


Abbildung 2.23: Beispiel-Objektstruktur für einen Produktentwicklungsprozeß

Anhand der in Kapitel 1.3 für Produktentwicklungsprozesse beschriebenen Objekthierarchie, läßt sich mit dem objektorientierten Datenmodell von WEP sehr einfach eine passende Objektstruktur bilden. Zusätzlich müssen nur noch die Qualitätsstufen modelliert werden. Abbildung 2.23 stellt die Objektstruktur dar, wie sie in WEP modelliert wird.

Das Beispiel zeigt alle Attribute, Relationen und Qualitätsstufen der jeweiligen Objektklassen. Dazu die Vererbungshierarchien der Klassen *Module* und *Part*. Die Relationen zeigen, daß eine *Kem-Klasse* aus *Dmu-Klassen* besteht. Diese wiederum bestehen aus Modulen, die elementar oder aus anderen Modulen zusammengesetzt sein können. *Module* ihrerseits bestehen aus *Parts* (Einzelteilen), die wiederum grundlegende oder zusammengesetzte Einzelteile sein können. Ein grundlegendes Einzelteil kann auf weitere Klassen, wie beispielswiese CAD-Daten verweisen, die die konkreten Inhalte für das Einzelteil enthalten.

Auf diese weitergehenden Daten gehen wir nicht weiter ein, da sie für die Betrachtung des Beispiels nicht von Belang sind. Abbildung 2.24 stellt eine mögliche Ausprägung für die in Abbildung 2.23 definierte Objektstruktur dar.

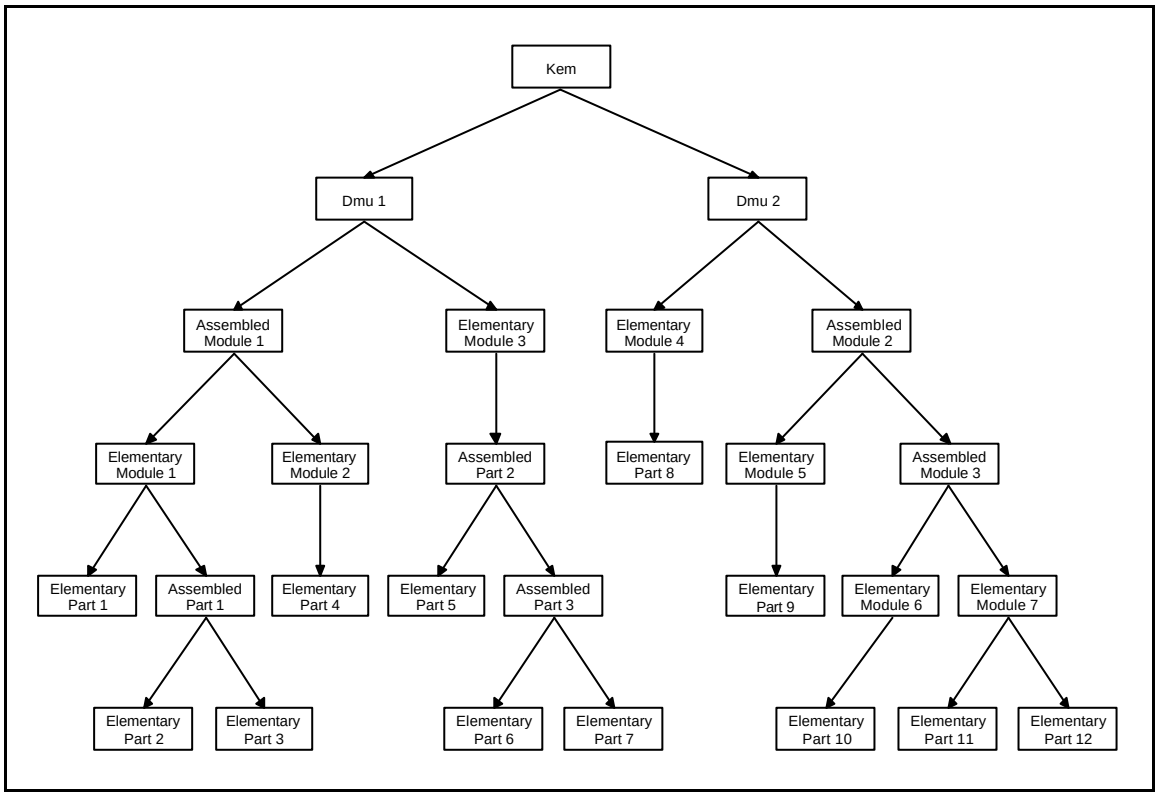


Abbildung 2.24: Beispiel-Ausprägung der Objektstruktur

Die Datenstrukturen der *ElementaryModuleClass* und der *SuperModuleClass* sehen anhand der Workflow-Beschreibungssprache aus, wie in Abbildung 2.25. Der Übersichtlichkeit halber wurden nicht alle in der Objektstruktur eingetragenen Qualitätsstufen beschrieben. Die fehlenden definieren sich entsprechend der dargestellten Qualitätsstufen.

```
 1   OBJECTCLASS SuperModuleClass
 2   {
 3      ATTRIBUTE state : NOTCHECKED | SUCCESS | NOSUCCESS
 4      ATTRIBUTE specification : STRING
 5      RELATIONSHIP consistsOfParts : SuperPartClass
 6      QUALITYLEVEL Spec
 7      QUALITYLEVEL VB
 8      ...
 9   }
10
11   OBJECTCLASS ElementaryModuleClass
12   {
13      SUPERTYPE SuperModuleClass
14      ATTRIBUTE report : STRING
15      QUALITYLEVEL Spec
16      {
17         EXPRESSION FORALL consistsOfParts QUALITYLEVEL == Spec
18      }
19      QUALITYLEVEL VB
20      {
21         EXPRESSION FORALL consistsOfParts QUALITYLEVEL == VB
22      }
23      ...
24   }
```

Abbildung 2.25: Beispiel-Workflow-Beschreibung einer Objektklasse

Die Zeilen 1 und 11 legen die Namen der Objektklassen fest. In Zeile 13 wird die Vererbung von *SuperModuleClass* nach *ElementaryModuleClass* festgelegt, das heißt, die Attribute, Beziehungen und Qualitätsstufen, die für *SuperModuleClass* definiert werden, existieren auch in *ElementaryModuleClass*. Zeile 3 definiert das Attribut *state* als Werteliste (*ValueList*). Die anderen Attribute sind Zeichenketten (*String*). Zeile 5 definiert eine Beziehung (*consistsOfParts*) auf die, in dem Modul enthaltenen Einzelteile. Die Zeile 6 und 7 deklarieren Qualitätsstufen. In der Subklasse können diese Qualitätsstufen verfeinert werden (Zeilen 15 bis 22).

Anhand einer, wie in Abbildung 2.24 gezeigten Ausprägung, muß die Feinplanung des Entwicklungsprozesses zur Laufzeit erfolgen. Sie zeigt schön die variable Anzahl von Unterobjekten in einer komplexen Objektstruktur, die zur Modellierungszeit nicht bekannt ist. Durch das Traversierungsmerkmal zur Unterstützung variabler Parallelität bietet das WEP-Modell hier das passende Konstrukt. Eine ‘übergeordnete’ Aktivität kann stellvertretend für die Entwicklung des gesamten Objekts gesehen werden (beispielweise ‘Entwickle Kem’). Traversierungsmerkmale splitten den Gesamtworkflow innerhalb dieser Pseudo-Aktivität in parallele Workflows auf. Zuerst geschieht dies auf der Ebene der Dmu-Klasse. Innerhalb dieser Traversierung kann weiter in die Anzahl der in der Dmu-Klasse enthaltener Module-Klassen gesplittet werden. Auf Basis der Module- / Part-Klassen setzt sich der Vorgang fort. Hier muß jedoch noch zwischen den verschiedenen Kategorien (Attribut *category*) der Part-Klasse (Einzelteil) unterschieden werden. Je nach Kategorie muß ein solches Einzelteil unterschiedlich entwickelt werden und benötigt somit auch eine eigene Aktivität für die Bearbeitung. Das Traversierungsmerkmal sollte also zusätzlich noch mittels eines booleschen Ausdrucks zwischen den möglichen Werten des Attributs *category* unterscheiden und erst dann auf die folgende Aktivität verweisen. Abbildung 2.26 zeigt die Workflow-Beschreibung für ein solches Traversierungsmerkmal.

```
 1   TRAVERSE SplitModul2Rohbauteil : ElementaryModuleClass
 2   {
 3      STARTACTIVITY EntwickleRohbauteil
 4      RELATION consistsOfParts
 5      {
 6         EXPRESSION state != SUCCESS
 7      }
 8      RELATION RECURSIVE consistsOfSubParts
 9      {
10         EXPRESSION state != SUCCESS
11         EXPRESSION category == ROHBAUTEIL
12      }
13   }
```

Abbildung 2.26: Beispiel-Workflow-Beschreibung für ein Traversierungsmerkmal

Die in einer Traversierung durchlaufenen Relationen werden in der Reihenfolge des Aufschreibens abgehandelt. Im ersten Schritt (Zeilen 4 bis 7) wird die Relation *consistsOfParts* der Klasse *ElementaryModuleClass* betrachtet. Die Traversierung erfolgt nur dann, wenn das Objekt der Klasse *SuperPartClass*, auf das durch die Relation verwiesen wird, noch nicht bereits als erfolgreich geprüft wurde. Dies wird durch den booleschen Ausdruck `state != SUCCESS` in Zeile 6 sichergestellt.

Im zweiten Schritt (Zeilen 8 bis 12) wird, nach erfolgreicher erster Traversierung, von einem Objekt der *Part*-Klasse ausgegangen. Dies kann, durch die in der Objektstruktur festgelegte Vererbungshierarchie, entweder eine *ElementaryPartClass* oder eine *AssembledPartClass* sein. Handelt es sich um ein zusammengesetztes Einzelteil (*AssembledPartClass*), so existiert die Relation *consistsOfSubParts*, die solange weiter rekursiv durchlaufen werden muß, bis man auf eine *ElementaryPartClass* stößt. An dieser Relation endet das rekusive Absteigen, da diese Klasse die Relation nicht mehr besitzt. Das gefundene Objekt dieser Klasse darf ebenfalls nicht als erfolgreich geprüft deklariert sein (Zeile 10) und muß der Kategorie Rohbauteil entsprechen (Zeile 11).

Wurde über die Relationen ein entsprechendes Objekt gefunden und treffen alle, über die booleschen Ausdrücke festgelegten Bedingungen zu, so wird ein neuer Workflow mit der Startaktivität *EntwickleRohbauteil* (Zeile 3) und dem gefundenen Einzelteil-Objekt als Eingabeobjekt gestartet.

Abbildung 2.27 zeigt den Traversierungssachverhalt anhand des Kontrollflusses. Die in der Abbildung dargestellten Einzelteil-Kategorien sind natürlich nur beispielhaft zu sehen. In der Realität gibt es wesentlich mehr Kategorien.

Der Gesamt-Workflow beginnt mit der Aktivität ‘Lege Konstruktionsumfang fest’. Hier wird die Objektstruktur, also das Kem-Objekt initialisiert. Danach folgt die oben bereits beschriebene Aufsplittung bis zur untersten Ebene der Einzelteile. Für jedes Einzelteil wird entsprechend seiner Kategorie, theoretisch ein neuer Workflow mit der passenden Aktivität gestartet, die die Entwicklung ausführt (Beispiel ‘Entwickle Rohbauteil’). Eine jeweils nachfolgende Prüfungs-Aktivität (Beispiel ‘Prüfe Rohbauteil’) testet die Korrektheit des entwickelten Einzelteils. Muß das Teil nachgebessert werden, findet per Schleifenkonstrukt ein Rücksprung zur Entwicklungs-Aktivität statt. Wie der Kontrollfluß innerhalb dieser Traversierung modelliert wird, zeigt Abbildung 2.28.

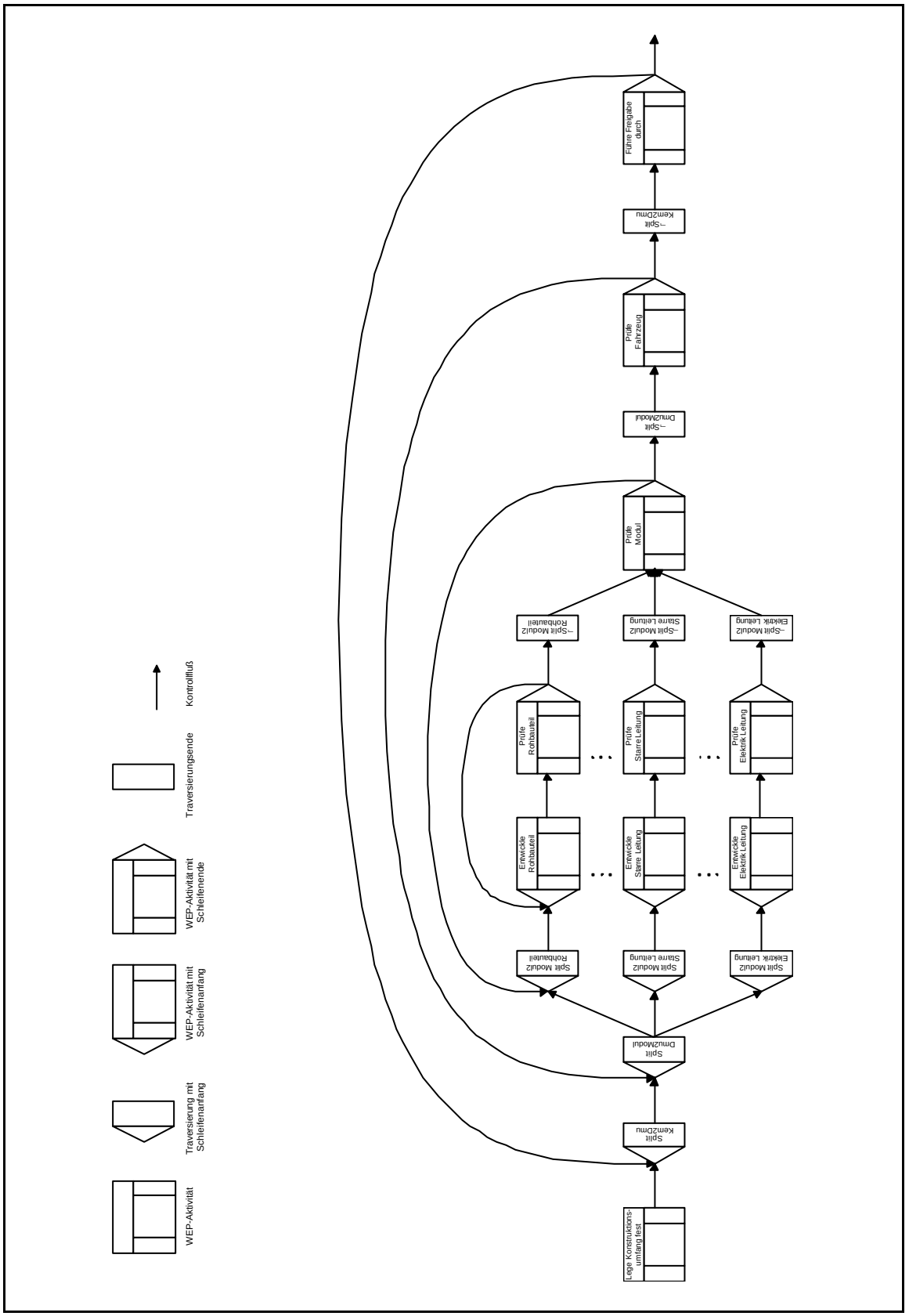


Abbildung 2.27: Kontrollfluß für den Beispiel-Workflow

```
 1   CONTROLFLOW EntwickleRohbauteilNachPrüfeRohbauteil
 2   {
 3      FROM EntwickleRohbauteil
 4      TO PrüfeRohbauteil
 5   }
 6
 7   CONTROLFLOW PrüfeRohbauteilOk
 8   {
 9      FROM PrüfeRohbauteil
10      TO TRAVERSEEND
11      IF RETURNCODE IS Erfolgreich
12   }
13
14   CONTROLFLOW PrüfeRohbauteilFail
15   {
16      FROM PrüfeRohbauteil
17      BACK TO EntwickleRohbauteil
18      IF RETURNCODE IS Nachbessern
19   }
```

Abbildung 2.28: Beispiel-Workflow-Beschreibung für den Kontrollfluß innerhalb der Einzelteil-Traversierung

Zuerst wird der Kontrollfluß zwischen der Entwicklung und der Prüfung beschrieben (Zeilen 1 bis 5). Dabei gibt es keine Besonderheiten. Von der Prüfungsaktivität gehen zwei Kontrollflußkanten aus. Wird bei der Prüfung ein Fehler entdeckt, so verweist die Kontrollflußkante *PrüfeRohbauteilFail* wieder zurück zur Entwicklung (Zeilen 14 bis 19). Wurde die Prüfung bestanden, kann die letzte Traversierung (Modul zu Einzelteil) wieder aufgehoben werden. Dies wird durch die Kontrollflußkante *PrüfeRohbauteilOK* dargestellt. Sie verweist nicht auf eine Aktivität, sondern auf das Traversierungsende (Zeile 10).

Nach der Prüfung auf Einzelteil-Ebene, bekommt die der Traversierung nachfolgende Aktivität ‘Prüfe Modul’ wieder den Zugriff auf die übergeordnete Modul-Klasse. Sie prüft, ob die im Modul zusammengefaßten Einzelteile zusammenpassen. Bei einem hier festgestellten Fehler muß bis zur letzten Traversierung zurückgesprungen werden, um wieder auf jedes Einzelteil einzeln zugreifen zu können und um sicherzustellen, daß eventuell neu hinzugekommene Einzelteile mitentwickelt werden. Derselbe Vorgang setzt sich mit den Ebenen der Dmu-Klasse (Aktivität ‘Prüfe Fahrzeug’) und Kem-Klasse (Aktivität ‘Führe Freigabe durch’) fort. Die letzte Aktivität des Workflows (‘Führe Freigabe durch’) gibt die Entwicklung quasi zur Produktion frei.

Abbildung 2.29 zeigt die Datenflußbeziehungen an dem eben beschriebenen Kontrollfluß. Die Struktur unter dem Kontrollfluß stellt die globalen Datenobjekte dar und zeigt, wie sie durch die Traversierungen von anderen globalen Datenobjekten abgeleitet werden. Die Datenflußbeziehungen beschreiben, welche Aktivitäten auf welche globalen Datenobjekte zugreifen. Die Datenflußbeziehungen der Einzelteil-Ebene sind nur für eine Einzelteil-Kategorie (‘Elektrik Leitung’) eingezeichnet.

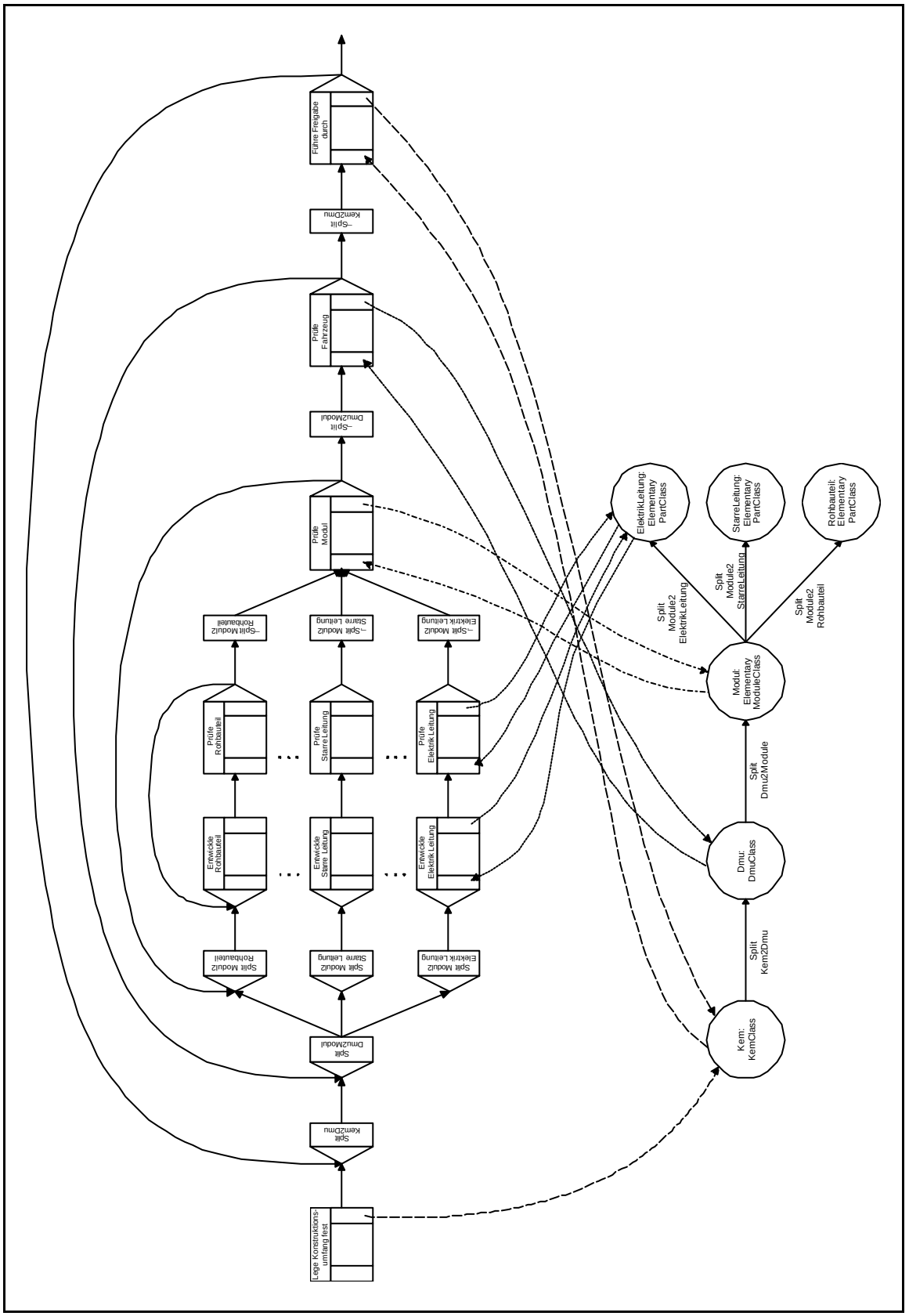


Abbildung 2.29: Kontrollfluß und Datenfluß für den Beispiel-Workflow

In Abbildung 2.30 wird die Definition eines globalen Datenobjekts und der ‘dazugehörenden’ Datenflußbeziehungen demonstriert. Das globale Datenobjekt *DmuObjekt* (Zeilen 1 bis 5) ist

aus der Klasse *DmuClass*. Über das Traversierungsmerkmal *SplitKem2Dmu* wird es vom globalen Datenobjekt *KemObjekt* abgeleitet (Zeile 4). Die dargestellten Datenflußkanten *PrüfeFahrzeugEingabe* (Zeilen 7 bis 12) und *PrüfeFahrzeugAusgabe* (Zeilen 14 bis 19) beschreiben die Datenflußbeziehungen zwischen dem Eingabe- und Ausgabeobjekt der Aktivität *PrüfeFahrzeug* und dem globalen Datenobjekt *DmuObjekt*.

```
 1   GLOBALOBJECT DmuObjekt : DmuClass
 2   {
 3      DBNAME DatenbankObjektDmu
 4      TRAVERSED OF KemObjekt.SplitKem2Dmu
 5   }
 6
 7   DATAFLOW PrüfeFahrzeugEingabe
 8   {
 9      TYPE IS INPUT
10      ACTIVITYOBJECT PrüfeFahrzeug.Dmu
11      GLOBALOBJECT DmuObjekt
12   }
13
14   DATAFLOW PrüfeFahrzeugAusgabe
15   {
16      TYPE IS OUTPUT
17      ACTIVITYOBJECT PrüfeFahrzeug.Dmu
18      GLOBALOBJECT DmuObjekt
19   }
```

Abbildung 2.30: Beispiel-Workflow-Beschreibung für den Datenfluß

Am Kontrollfluß ist gut zu erkennen, wie die zielorientierten Aktivitäten des WEP-Modells durch die vorzeitige Datenweitergabe eine Beschleunigung des Entwicklungsprozesses bewirken können. Ab den Entwicklungsaktivitäten können vorzeitige Daten für die Prüfung bis hinauf auf Kem-Ebene weitergegeben werden. Die Prüfungen können so eventuell schon bei Datenobjekten mit niedrigerer Qualitätsstufe feststellen, daß bei der Entwicklung nachgebessert werden muß. Dadurch kann die Entwicklung bereits nach kurzer Zeit angepaßt werden, ohne erst das gesamte Einzelteil entwickeln zu müssen.

Um die vorzeitige Datenweitergabe möglich zu machen, müssen in den zielorientierten Aktivitäten natürlich die erforderlichen Eingabeobjekte und die zu erstellenden Ausgabeobjekte, also die zu erreichenden Ziele (Meilensteine) definiert werden. Für die Aktivität *EntwickleRohbauteil* beispielsweise, muß als Eingabeobjekt das zu entwickelnde Einzelteil mit der Entwicklungsspezifikation vorliegen. Über die Meilensteine werden die verschiedenen Qualitätsstufen des Einzelteils definiert und beschrieben, wann sie bereitstehen müssen. Wie die Meilensteine für diese Aktivität spezifiziert werden, zeigt Abbildung 2.31. Zur Erklärung des Beispiels reicht die Darstellung von zwei Meilensteinen. Die Spezifikation der anderen Meilensteine ist entsprechend den dargestellten.

```
 1   ACTIVITY EntwickleRohbauteil
 2   {
 3      PROCESSED BY Konstrukteur
 4      IN REQUIRED Einzelteil : ElementaryPartClass
 5      {
 6         QUALITYRANGE Spec, VB, GG, FG, VS
 7         CONSISTENCYPOLICY MERGE
 8      }
 9      OUT Einzelteil : ElementaryPartClass
10      {
11         REQUIRED IN QUALITYLEVEL EntwickleRohbauteil_VB
12         {
13            BASED ON OBJECT QUALITYLEVEL VB
14            PREREQUISITE Einzelteil.Spec
15            TIMELIMIT 10 DAYS
16            CONCURRENTMODE AUTONOMOUS
17         }
18         ...
19         REQUIRED IN QUALITYLEVEL EntwickleRohbauteil_VS
20         {
21            BASED ON OBJECT QUALITYLEVEL VS
22            PREREQUISITE Einzelteil.Spec
23            TIMELIMIT 40 DAYS
24            CONCURRENTMODE AUTONOMOUS
25         }
26      }
27      PROGRAMSTEP Erstelle2DZeichnung
28      ...
29      RETURNCODE Konstruiert
30      {
31         REQUIRED Einzelteil.EntwickleRohbauteil_VS
32      }
33   }
```

Abbildung 2.31: Beispiel-Workflow-Beschreibung für eine zielorientierte Aktivität

Für die Aktivität wird zuerst festgelegt, von welchem Bearbeiter sie ausgeführt werden kann (Zeile 3). Dann wird das Eingabeobjekt *Einzelteil* mit den erlaubten Qualitätsstufen modelliert (Zeile 6), sowie die Strategie festgelegt, die bei der Integration einer neuen Objektversion angewandt wird (Zeile 7). Im Falle einer Entwicklungs-Aktivität ist es sicher sinnvoller, eine neue Objektversion der Vorgänger-Aktivität mit der eigenen Objektversion abzugleichen, da ein Undo für eine möglicherweise langdauernde Entwicklung natürlich sehr zeitaufwendig ist. Das Ausgabeobjekt ist genau das gleiche wie das Eingabeobjekt, nur eben mit den neuen Entwicklungsdaten versehen (Zeilen 9 bis 26). Laut der Objektstruktur hat ein Objekt der Klasse *ElementaryPartClass* die fünf Qualitätsstufen Spec, VB, GG, FG und VS. Für jede der vier höheren Qualitätsstufen muß das Objekt weiterentwickelt werden und benötigt somit die Definition eines Meilensteins. Der erste Meilenstein in der Abbildung basiert auf der Qualitätsstufe *VB* (Zeilen 11 bis 17). Das Merkmal *Prerequisite* legt fest, welche Qualitätsstufe als Vorraussetzung für diese Qualitätsstufe mindestens bereits existieren muß (Zeile 14) Dies ist für eine Entwicklungs-Aktivität natürlich die Spezifikation des Objekts (*Spec*). Das Merkmal *Timelimit* gibt einen logischen Zeitpunkt an, zu dem diese Qualität ab Aktivitätsstart zur Verfügung stehen muß (Zeile 15). Die Angabe bedeutet, daß die Qualitätsstufe *VB* 10 Tage nach dem Start der Aktivität bereitstehen muß. Weiter wird noch festgelegt, welche Strategie

bei der Weitergabe neuer Versionen dieses Objekts verfolgt wird (Zeile 16). Im Falle der Entwicklung kann die Objektversion freigegeben werden, ohne die Zustimmung der abhängigen Prüfungs-Aktivitäten. Weitere Angaben betreffen die verfügbaren Programmschritte (Zeile 27) und die Rückgabewerte der Aktivität (Zeile 29 bis 32). Eine Entwicklungs-Aktivität besitzt nur einen Rückgabewert. Er gibt an, daß die höchste Qualitätsstufe des Ausgabeobjekts (*VS*) erreicht ist. Damit ist die Entwicklung beendet.

2.10 Bewertung

Zum Schluß betrachten wir noch einmal alle Eigenschaften von WEP und versuchen zu bewerten, wie gut sich das Modell damit für die Unterstützung von Produktentwicklungsprozessen eignet.

Aufgrund des dem WEP-Modell zugrundeliegenden prozeßorientierten Modells und der Erweiterung der variablen Parallelität, entsteht der Kontrollfluß eines WEP-Workflows eigentlich immer nach dem gleichen Schema. Abbildung 2.27 zeigt ein Beispiel eines solchen Kontrollflusses, der durch die symmetrische Blockstrukturierung charakterisiert ist. Diese Kontrollflußstruktur stellt die Grobplanung einer Produktentwicklung anhand der Organisationsstruktur des Unternehmens dar. Durch die Traversierungsaktivitäten läßt sich anhand der zugrundeliegenden Datenstruktur die Feinplanung entwickeln. Dieses Vorgehen ist für Entwicklungsprozesse sehr gut geeignet, da sich die Organisationsstruktur eines Unternehmens und damit die Grobplanung eines Prozeßablaufs nur selten ändert. Dies ist bei der Feinplanung anders. Sie orientiert sich immer an aktuellen Gegebenheiten und kann bei jedem Entwicklungsprozeß verschieden sein.

Eine einfache Anpassung eines gesamten WEP-Workflows an Änderungen, um schneller auf Kundenwünsche und Innovationen reagieren zu können, läßt sich dennoch nicht so einfach realisieren. Dies ist durch die Trennung von Modellierungs- (Build-time) und Laufzeitkomponente (Run-time) bedingt. Eine Änderung des Workflows oder eine Neuorganisation der Datenstruktur erfordert auf jeden Fall einen Stopp der aktuellen Ausführung und eine Neumodellierung über eine neue Workflow-Beschreibung. Erstreckt sich die Änderung jedoch nur auf die Datenstruktur, so ist eine Anpassung nicht erforderlich.

Sehr flexibel ist WEP in der Unterstützung unstrukturierter kreativer Teilprozesse. Zielorientierte Aktivitäten sind hier ein guter Ansatz. Sie bieten dem Bearbeiter genügend Möglichkeiten zur freien Entfaltung seiner Kreativität und eigener Arbeitsweisen in der Entwicklungsphase. Durch die Festlegung von Meilensteinen als Ziele einer Aktivität, hat er genaue Vorgaben, wann er welche Daten abzuliefern hat. Die möglichen Werkzeuge für seine Arbeit werden ihm als Programmschritte innerhalb der zielorientierten Aktivität zur Verfügung gestellt. Über seine Arbeitsreihenfolge, das heißt, in welcher Reihenfolge er die Programmschritte durchführt, kann er somit frei entscheiden.

Die Definition von Meilensteinen impliziert Zeitangaben. Damit realisiert WEP ein Zeitmanagement, das für die Einhaltung von Fristen und Fertigungsterminen sorgt. Der Bearbeiter erhält darüberhinaus eine Benachrichtigung vom System, sollte er den festgelegten Zeitpunkt eines Meilensteins nicht eingehalten haben.

Die zusätzliche Einbringung von Datenqualitäten in die Beschreibung der Meilensteine ermöglicht bei WEP die direkte Unterstützung von simultanen Arbeitsschritten (Simultaneous Engineering). Gibt eine Aktivität ein Datenobjekt mit vorläufiger Datenqualität weiter, so sind nachfolgende

Aktivitäten bereits vor Erreichen der endgültigen Daten in der Lage mit ihrer Arbeit zu beginnen und können damit zur Verkürzung der Entwicklungszeiten beitragen. Dadurch können auch früher Fehler erkannt und an die entsprechenden Entwickler zurückgemeldet werden, was im Fehlerfall zu einer Verringerung von Wiederholungen in der Entwicklung führt. Bei Fehlern in den erstellten Datenobjekten oder bei Mißbilligung einer neuen Objektversion, bringt ein synchroner Gruppenarbeitsmechanismus (Konsolidierungsrunde) eine zusätzliche Unterstützung für den Datenaustausch und die Kommunikation zwischen den Bearbeitern.

Aufgrund der im Entwicklungsbereich üblichen komplexen Datenstrukturen muß ein geeignetes System auch dem zugrundeliegenden Datenmodell entsprechende Beachtung schenken. WEP verwendet ein objektorientiertes Modell, das dafür die zweckmäßigste Darstellungsform ist. Neben der üblichen Unterstützung von Klassen, Relationen, Attributen und Vererbung, erweitert WEP das objektorientierte Modell noch durch die Angabe von Qualitätsstufen. Diese Erweiterung ist ausschlaggebend für die vorzeitige Datenweitergabe. Die Festlegung der Datenqualitäten eines Ausgabeobjekts einer zielorientierten Aktivität basiert auf den Qualitätsstufen der zugrundeliegenden Objektstruktur.

Das WEP-Modell bringt viele Eigenschaften mit, die besonders an Produktentwicklungsprozesse angepaßt sind. Es unterstützt damit so gut wie alle gestellten Anforderungen. Daraus resultiert jedoch auch eine große Komplexität auf der Ebene der Workflowverwaltung. Das System muß ständig die Zeitangaben und Datenqualitäten der Meilensteine überprüfen. Darüberhinaus bedingt die variable Anzahl paralleler Kontrollflüsse eine wesentlich komplexere Abhängigkeitsverwaltung als bei herkömmlichen Systemen.

Die zusätzlichen Interaktionsformen und die Überwachung der Meilensteine erhöhen natürlich auch den Implementierungsaufwand. Dennoch ist auf eine effiziente Implementierung zu achten, da ein WEP-Workflow-Management-System trotz der komplexen Abhängigkeiten benutzertolerierbare Antwortzeiten liefern muß.

3 Abgrenzung

In diesem Kapitel werden mehrere Ansätze aus verwandten Gebieten beschrieben (Abbildung 3.1), die sich zum Teil mit WEP überschneiden oder aber ganz andere Wege verfolgen. Alle betrachteten Ansätze haben jedoch eines gemein. Sie versuchen die Planung, Verwaltung und Kooperation bei komplexen Arbeitsabläufen flexibler zu unterstützen und zu integrieren als dies bisher mit den konventionellen Systemen des Workflow-Managements, des Projekt-Managements und der Groupware möglich ist.

Mit den folgenden Kapiteln werden einige ausgewählte Ansätze detaillierter besprochen. Im Anschluß an die konzeptionelle Beschreibung eines jeden Modells möchte ich versuchen, das in Kapitel 2 bereits bei WEP beschriebene Beispiel in das entsprechende Modell umzusetzen. Abschließend erfolgt jeweils eine Abgrenzung und Bewertung im Vergleich zum WEP-Modell.

Am Ende dieses Kapitels soll eine tabellarische Zusammenfassung einen genauen Aufschluß darüber geben, welche Eigenschaften der betrachteten Ansätze den Anforderungen des WEP-Modells am nächsten kommen.

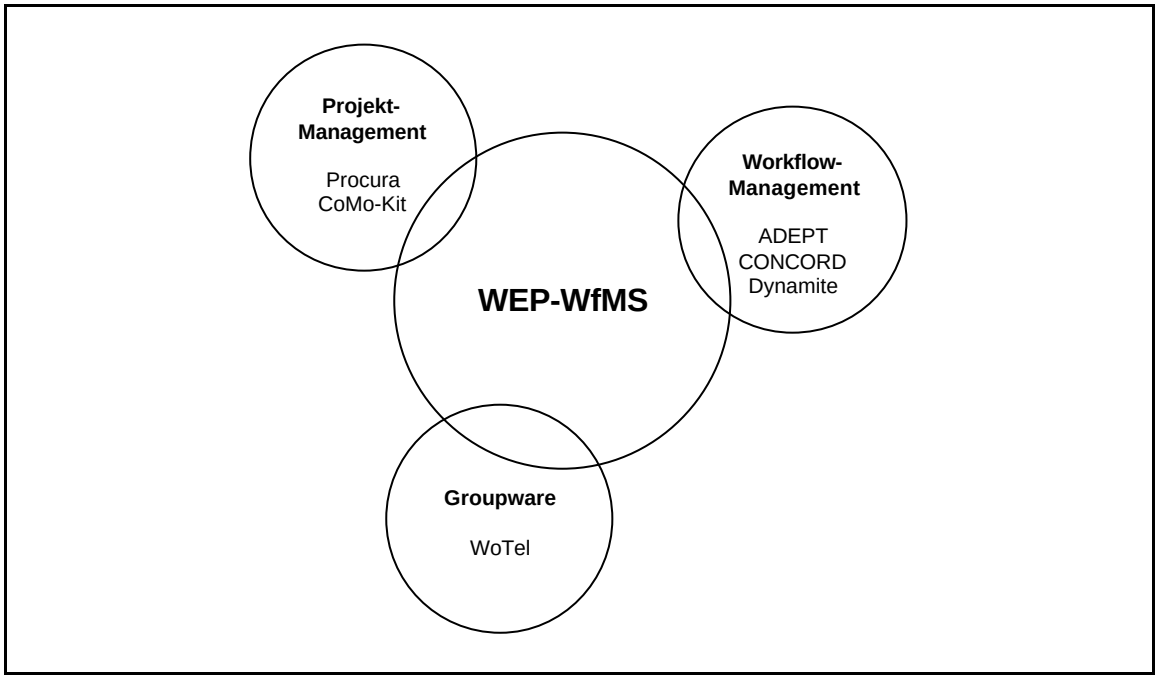


Abbildung 3.1: Abgrenzungen zum WEP-Modell

Drei der beschriebenen Ansätze passen am besten in das Gebiet des Workflow-Managements. Dazu gehören die Systeme **ADEPT** (Kapitel 3.1), **CONCORD** (Kapitel 3.2) und **Dynamite** (Kapitel 3.3). Eher dem Bereich des Projekt-Managements zuordnen kann man die Systeme **Procura** (Kapitel 3.4) und **CoMo-Kit** (Kapitel 3.5). Daran anschließend folgt eine Betrachtung des **WoTel-Projekts** (Kapitel 3.6). WoTel nimmt unter den beschriebenen Ansätzen eine Sonderstellung ein, da es sich dabei um kein System handelt, das in der Lage ist Arbeitsabläufe zu verwalten. WoTel läßt sich am einfachsten als Groupware-Applikation bezeichnen und wurde hinzugezogen, um einen Vergleich mit den Kommunikationsansätzen der WEP-Konsolidierungsphase ziehen zu können.

3.1 ADEPT

Als erstes Beispiel werden wir ADEPT betrachten. ADEPT steht für **A**pplication **DE**velopment Based on **P**re-Modeled Activity **T**emplates. Die Idee dazu entstand anfangs im Rahmen des OKIS[1]-Projekts zur Unterstützung und Entwicklung flexibler und zuverlässig kooperierender Assistenzsysteme in klinischen Anwendungsumgebungen (vgl. [DKR+95]). Ein besonderes Konzept, mit dem sich ADEPT dabei von anderen Workflow-Modellen abhebt, ist seine Flexibilität, das heißt kontrolliertes Einfügen und Löschen von Arbeitsschritten.

Um den Entwicklungsaufwand bei der geforderten Zuverlässigkeit und Flexibilität in Grenzen zu halten, muß man sich auf Teilaspekte konzentrieren, die damit überschaubar bleiben. An diesem Punkt setzt ADEPT an. Es zerlegt die Gesamtaufgabe in Teilaufgaben und reduziert somit die Komplexität. Dieses Vorgehen ist in 5 Phasen unterteilt:

In der ersten Phase wird ein Ablaufmodell (Activity Template) erstellt oder ausgewählt. Dieses Modell beschreibt wie bei prozeßorientierten Workflow-Management-Systemen mittels Kanten, Verzweigungen und Knoten den Kontroll- und Datenfluß.

In Phase zwei wird das Ablaufmodell durch bedingt wählbare Aktionen (Ausnahmen), Zeitrestriktionen, vorgeplante Ausnahmeaktionen (Überspringen, Shortcuts) und Rücksprünge (Wiederholung, Stornierung) ergänzt.

Die dritte Phase fügt Kompensationssphären ein. Eine Kompensationssphäre besagt, daß nach Beginn eines außerhalb der Sphäre liegenden Teilschrittes kein Teilschritt innerhalb der Sphäre mehr kompensiert, also abgebrochen und rückgängig gemacht werden darf. Nach diesen ersten drei Schritten ist die Ablauflogik festgelegt. Damit kann die Ausführung und Behandlung von Ausnahmefällen systemseitig überwacht und bearbeitet werden.

In der vierten Phase werden die einzelnen Teilschritte als eigenständige Programme oder als Schnittstellen zu bestehenden Programmen implementiert. Diese Programmbausteine sind bis auf die Parameterversorgung, die vom Laufzeitsystem abhängig ist, vollständig eigenständig. Dies erleichtert nicht nur die Programmierung, zum Beispiel durch Outsourcing, sondern ermöglicht außerdem von allen anderen Komponenten unabhängiges isoliertes Testen.

Nach der Implementierung aller Bausteine, werden diese in Phase fünf in das Activity Template eingesetzt, wodurch es zur Aktivitätenimplementierung wird. Desweiteren werden die Input- und Output-Parameter geeignet verbunden. Nach Beendigung dieser fünf Phasen ist die Aktivitätenimplementierung - nach außen hin - ein ablauffähiges Programm. Dieses Programm kann mittels Standardmethoden angesprochen werden. Die Standardmethoden müssen von einer Laufzeitumgebung (Activity Shell) zur Verfügung gestellt werden.

3.1.1 Grundlagen des Modells

Die Basis für einen Workflow in ADEPT ist ein graphenorientiertes Modell, das in seiner Syntax und Semantik auf einer formalen Grundlage beruht. Essentiell für die Modellierung und Ausführung eines Workflows ist dabei das Konzept der symmetrischen Kontrollstrukturen (vgl. [ReDa97]). So wird ein Workflow als eine Aneinanderreihung symmetrischer Blöcke mit

[1] OKIS - Offenes klinisches Datenbank- und Informationssystem zur Integration autonomer Subsysteme

Sequenzen, Verzweigungen und Schleifen und mit eindeutigem Start- und Endknoten definiert. Diese Blöcke können beliebig verschachtelt werden, dürfen sich aber nicht überlappen. Zusätzlich unterstützt ADEPT die Synchronisation verschiedener Zweige innerhalb eines solchen Ablaufmodells.

ADEPT modelliert neben dem Kontrollfluß eines Workflows auch den Datenfluß. Da dadurch auch die Abhängigkeiten zwischen den Aktivitäten des Workflows und den benötigten Datenwerten formal geregelt werden, besteht eine wesentlich bessere Grundlage für die dynamische Änderung eines Workflow-Modells. Diese dynamischen Änderungen sind ein generelles Ziel von ADEPT, weshalb wir später noch näher darauf eingehen werden.

Wir werden zur einfacheren Betrachtung der Modellierung die grafische Darstellung verwenden, zur Betrachtung der formalen Definitionen wird auf [ReDa98] verwiesen.

3.1.2 Kontrollfluß

Der Kontrollfluß eines Workflow-Schemas in ADEPT wird als gerichteter strukturierter Graph dargestellt. Aktivitäten werden durch Knoten und Abhängigkeiten zwischen diesen Knoten durch Kanten repräsentiert. Knoten und Kanten gibt es in mehreren verschiedenen Typen. Jedes Workflow-Schema hat dabei jeweils einen eindeutigen Start- und Endknoten.

Für die Modellierung des Kontrollflusses stehen mehrere grundsätzliche Konstrukte zur Verfügung. Am einfachsten ist die sequentielle Ausführung. Bei einer **Sequenz** (Abbildung 3.2) hat jede Aktivität einen Nachfolger, der über eine Kontrollkante mit dem Vorgänger verbunden ist.

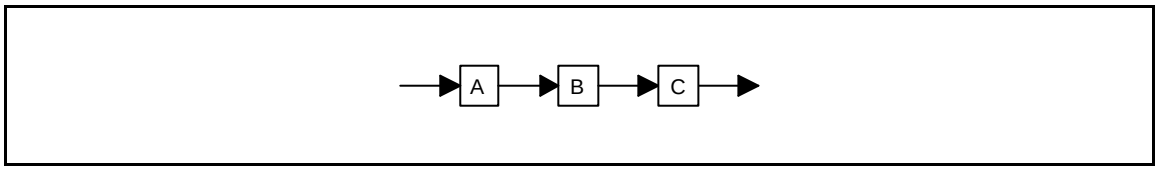


Abbildung 3.2: Sequenz in ADEPT

Verzweigungen (Abbildung 3.3) beginnen mit einem Verzweigungsknoten und enden, durch die symmetrische Blockstrukturierung, in einem eindeutigen Vereinigungsknoten. Es gibt drei Arten von Verzweigungen:

a) **Parallele Verzweigung**: Die parallele Verzweigung beginnt mit einer UND-Verzweigung und endet in einer UND-Vereinigung. Alle zwischen diesen Knoten liegenden Zweige werden parallel gestartet und erst nach Beendigung aller kann mit dem Vereinigungsknoten fortgefahren werden.

b) **Bedingte Verzweigung**: Die bedingte Verzweigung beginnt mit einer ODER-Verzweigung und endet mit einer ODER-Vereinigung. Von den dazwischenliegenden Zweigen wird nur einer gestartet und durchgeführt. Die Wahl des zu bearbeitenden Zweiges kann explizit im Verzweigungsknoten auf Basis eines Datenwertes geschehen oder implizit durch Auswahl des Benutzers selektiert werden.

c) **Parallele Verzweigung mit finaler Auswahl**: Dieses Konstrukt beginnt mit einer UND-Verzweigung und endet in einer ODER-Vereinigung. Alle Zweige werden parallel gestartet.

Der Zweig der zuerst beendet wird, bestimmt mit seinen Informationen die Fortführung der nachfolgenden Aktivitäten. Die anderen Zweige werden zurückgesetzt.

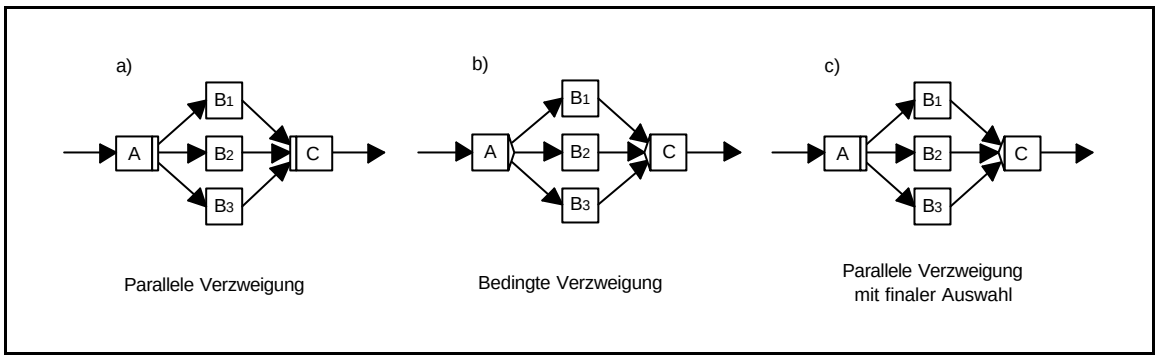


Abbildung 3.3: Verzweigungen in ADEPT

Zur Unterstützung von zyklischen Graphen gibt es noch das **Schleifenkonstrukt** (Abbildung 3.4). Analog zu den anderen Konstrukten ist eine Schleife ein symmetrischer Block mit einem eindeutigen Startknoten und einem eindeutigen Endknoten. Der Endknoten ist über eine Schleifenkante mit dem Startknoten verbunden. Desweiteren ist der Endknoten mit einer Bedingung verknüpft, die nach jedem Schleifendurchlauf ausgewertet wird und damit den weiteren Ablauf bestimmt. Entweder wird die Schleife verlassen oder über die Schleifenkante erneut durchlaufen.

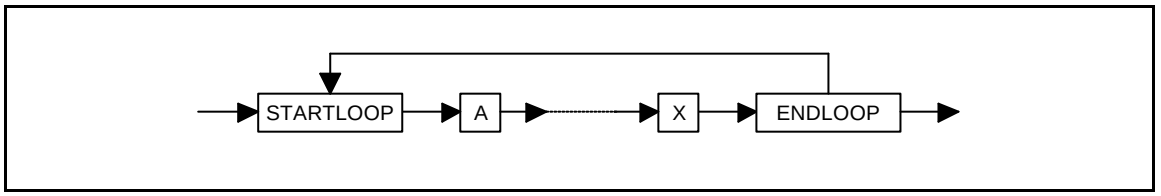


Abbildung 3.4: Schleife in ADEPT

Für die Synchronisation von Aktivitäten unterschiedlicher Ausführungszweige, zwischen denen Datenabhängigkeiten bestehen, gibt es zwei spezielle **Synchronisationskanten** (Abbildung 3.5):

- **STRICT_SYNC**: Die strikte oder harte Synchronisationskante von Aktivität A nach Aktivität B bedingt, daß A erfolgreich abgeschlossen sein muß, bevor B gestartet werden kann.
- **SOFT_SYNC**: Die schwache oder weiche Synchronisationskante von Aktivität A nach Aktivität B bedingt eine etwas schwächere Synchronisationsbedingung. Hier kann B nur gestartet werden, wenn A entweder erfolgreich beendet wurde oder nicht mehr zur Ausführung kommt (z.B. wenn A Teil eines ausgelassenen Zweiges einer Bedingten Verzweigung ist).

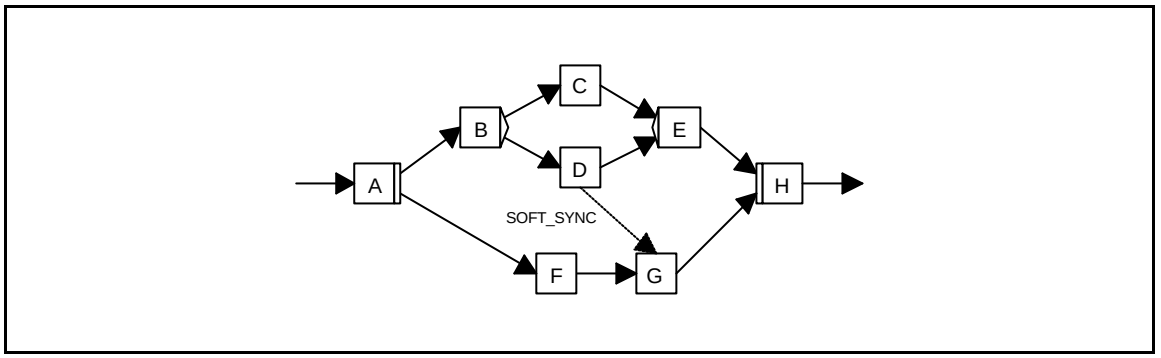


Abbildung 3.5: Synchronisationskanten in ADEPT

Um redundante Abhängigkeiten, Zyklen und nicht erreichbare Knoten im Graphen zu vermeiden muß die Benutzung der Synchronisationskanten gewissen Einschränkungen unterliegen. Auf jeden Fall dürfen Synchronisationskanten nur zwischen Aktivitäten aus Zweigen paralleler Verzweigungen bestehen und z.B. nicht von einer Aktivität außerhalb einer Schleife auf eine Aktivität innerhalb der Schleife verweisen.

3.1.3 Datenfluß

ADEPT modelliert neben dem Kontrollfluß auch noch den Datenfluß (Abbildung 3.6), also die Abhängigkeiten zwischen Datenwerten (Variablen) und Aktivitäten eines Workflow-Schemas.

Jedem Workflow-Schema wird dabei ein Satz von Variablen (Datensatz) zugewiesen. Der Datenfluß zwischen den Aktivitäten des Workflows wird nun dadurch definiert, daß die Eingabe- bzw. Ausgabe-Parameter einer Aktivität mit einzelnen Variablen aus dem Datensatz verbunden werden. Durch das Graphenkonzept bedingt sind nun die Eingabedaten des Workflows die Ausgabedaten des Startknotens des Workflows.

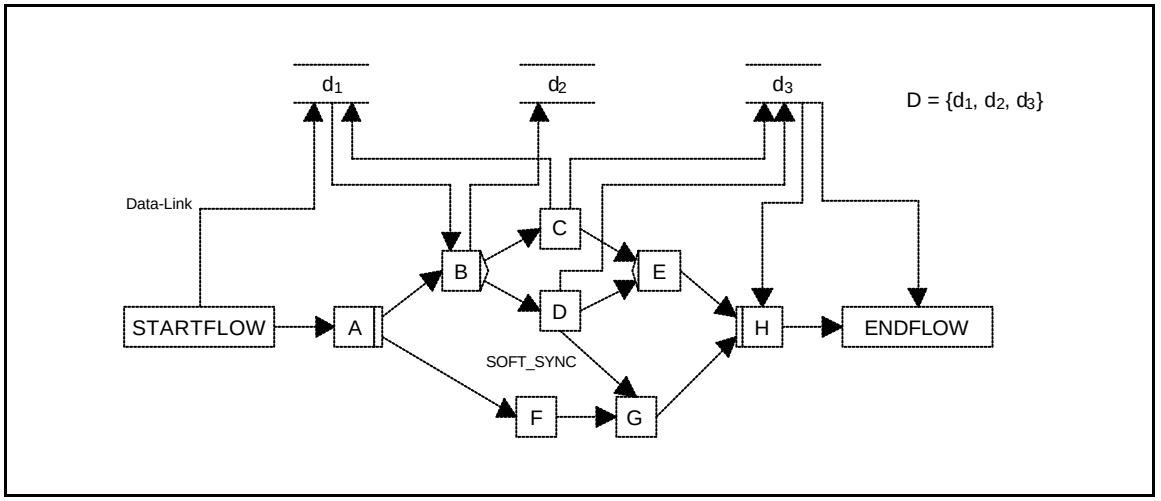


Abbildung 3.6: Datenfluß in ADEPT

Das Datenflußmodell verbindet die Eingabe- und Ausgabeparameter jeder Aktivität per Datenkante (Data-Link) mit den entsprechenden Variablen des Datensatzes. Fungiert eine

Variable als Eingabeparameter so verweist ein Link auf den entsprechenden Knoten. Ein Link mit umgedrehter Richtung vom Knoten aus verweist dagegen auf einen Ausgabeparameter.

Für dieses Vorgehen müssen gewisse Regeln gelten. So gehen wir davon aus, daß vor dem Beginn einer Aktivität alle benötigten Eingabewerte existieren (d.h. von einer vorausgehenden Aktivität erzeugt wurden) und nach erfolgreicher Beendigung der Aktivität alle Ausgabewerte gesetzt sind. Um Fehler in der Modellierung zu vermeiden, sollte deshalb darauf geachtet werden, daß jeder Eingabe- und Ausgabeparameter mit exakt einem Data-Link verbunden ist. Desweiteren müssen über Data-Links verbundene Variablen und Parameter natürlich dem gleichen Datentyp entsprechen. Zur Vermeidung von Lost Updates dürfen außerdem Aktivitäten aus parallelen Ausführungszweigen (parallele Verzweigung) keinen Schreibzugriff auf dieselben Variablen haben, es sei denn sie sind über eine Synchronisationskante passend verbunden.

Trotz aller Vorsicht bleibt es bei manchen dynamischen Änderungen nicht aus, daß die vorgegebenen Regeln verletzt werden. So kann das Löschen einer Aktivität, das Löschen von Data-Links verursachen, wodurch nachfolgenden Aktivitäten Parameter fehlen. Andererseits kann das Einfügen einer Aktivität mit dem Einfügen verbundener Data-Links wiederum ein Lost Update-Problem entstehen lassen. Auf diese Problematik wird nochmals bei der Behandlung dynamischer Änderungen eingegangen.

3.1.4 Workflowausführung

Der Status eines in der Ausführung befindlichen Workflows ergibt sich aus dem Status aller enthaltenen Knoten (Aktivitäten) und Kanten (Beziehungen zwischen den Aktivitäten), den Werten der benutzten Variablen und Informationen über den bisherigen Ablauf der Ausführung.

Der Status eines Knotens kann sein: NOT_ACTIVATED, ACTIVATED, RUNNING, COMPLETED oder SKIPPED. Der Status einer Kante ist entweder NOT_SIGNALED, FALSE_SIGNALED oder TRUE_SIGNALED. Jedesmal wenn eine Kante einen neuen Status signalisiert, wird der Status des Zielknotens entsprechend den von ADEPT festgelegten Ausführungsregeln neu berechnet. Diese Regeln bestimmen, wann und wie ein Knoten seinen Status ändert.

Betrachten wir als Beispiel einen UND-Vereinigungsknoten mit dem Status NOT_ACTIVATED (Abbildung 3.7). Zuerst haben alle eingehenden Kontrollkanten den Status NOT_SIGNALED, ebenso die ausgehende Kontrollkante (a). Sobald alle eingehenden Kontrollkanten den Status TRUE_SIGNALED setzen, ändert der Knoten seinen Status auf ACTIVATED. Wird die Aktivität des Knotens ausgeführt gelangt er in den Status RUNNING (b). Nach seiner erfolgreichen Beendigung (Status COMPLETED) wird die von ihm ausgehende Kontrollkante von NOT_SIGNALED auf TRUE_SIGNALED geändert (c).

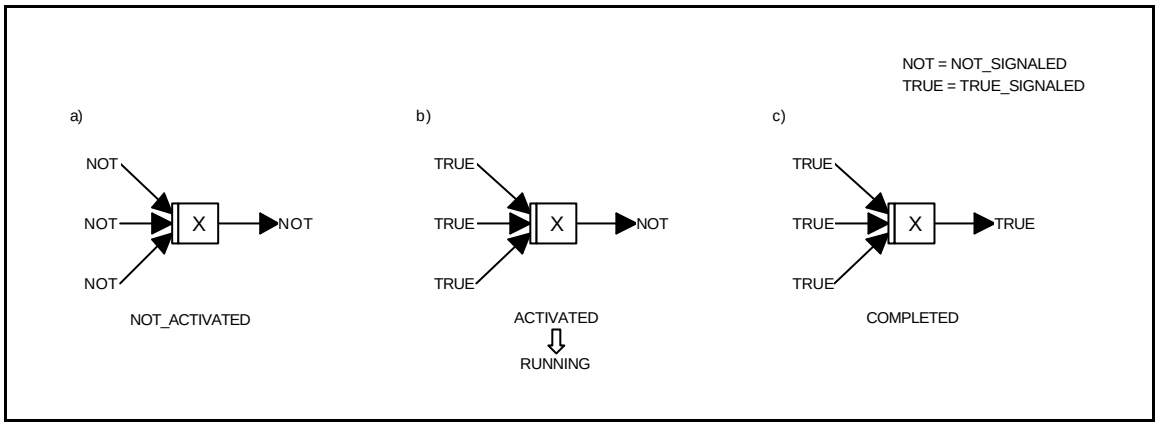


Abbildung 3.7: Workflowausführung in ADEPT

Wird die Abarbeitung der Aktivität des Knotens abgebrochen, ändert die ausgehenden Kante ihren Status in FALSE_SIGNALED. Dies führt dann eventuell zum Auslassen der Aktivitäten folgender Knoten.

3.1.5 Dynamische Änderungen ($ADEPT_{flex}$)

Alle Konzepte, die ADEPT zur Modellierung des Kontroll- und Datenflusses einsetzt, sind speziell auf das Ziel ausgerichtet, dynamische Änderungen am Workflow-Modell durchzuführen. Zu jeder Zeit und möglichst ohne den Verlust bereits ausgeführter Aktivitäten und deren Daten. Zur Erreichung dieses Ziels trägt hauptsächlich die starke formelle Ausrichtung des ADEPT-Modells bei.

Andere Workflow-Management-Systeme erlauben oftmals überhaupt keine dynamische Änderung an einem Workflow. Falls dynamische Änderungen erlaubt werden, so werden eigentlich wichtige Aspekte, wie die Datenintegrität, meist außer Acht gelassen. Änderungen werden auf solche Varianten beschränkt, die problemlos zu handhaben sind, oder werden nur zugelassen, wenn ein Workflow inaktiv ist.

Aufbauend auf ADEPT wurden unter dem Namen $ADEPT_{flex}$ (vgl. [ReDa98]) einige Methoden und Vorgehensweisen zur Unterstützung der dynamischen Änderungen an einem Workflow-Modell zusammengefaßt. Das Hauptaugenmerk wurde auf die Konsistenzhaltung und Korrektheit eines geänderten Workflows gelegt. Eine Änderung muß also wieder ein korrektes Workflow-Schema mit einem legalen Status hinterlassen. Dabei ist jede Änderungsvariante und dies in jedem Zustand, in dem sich ein Workflow gerade befindet, zugelassen.

$ADEPT_{flex}$ bietet spezielle Unterstützung für das Einfügen und Löschen von Aktivitäten, respektive Knoten, aus einem Workflow-Schema. Daß dabei der Workflow konsistent und korrekt gehalten wird, wird durch verschiedene Vorgehensweisen erreicht. Dazu zählt das Überspringen von Aktivitäten, das Serialisieren von vorher parallel ausgeführten Zweigen oder das dynamische Wiederholen bzw. Zurücknehmen der Ausführung eines Teils des Workflows.

Das generelle Vorgehen einer Änderungsaktion kann man in vier Punkte untergliedern: Festlegen der Vorbedingungen, Änderungsalgorithmus, Neubewertung der Zustände des Workflows und Anpassen der Datenflußbindungen.

Anhand eines Beispiels, nämlich das Einfügen eines Knotens, wollen wir $ADEPT_{flex}$ näher betrachten (Abbildung 3.8 bis Abbildung 3.11). Wir legen fest, daß der Knoten X in das Workflow-Modell zwischen die Knoten M_{before} = {C, D} auf der einen Seite und den Knoten M_{after} = {F} auf der anderen Seite eingefügt werden soll. Die Aktivität X soll also aktiviert werden, sobald für die Aktivitäten C und D jeweils gilt: Die Aktivität wurde beendet oder es steht fest, daß sie nicht mehr bearbeitet wird. Die Aktivität F kann dagegen erst aktiviert werden, wenn X beendet ist (Abbildung 3.8).

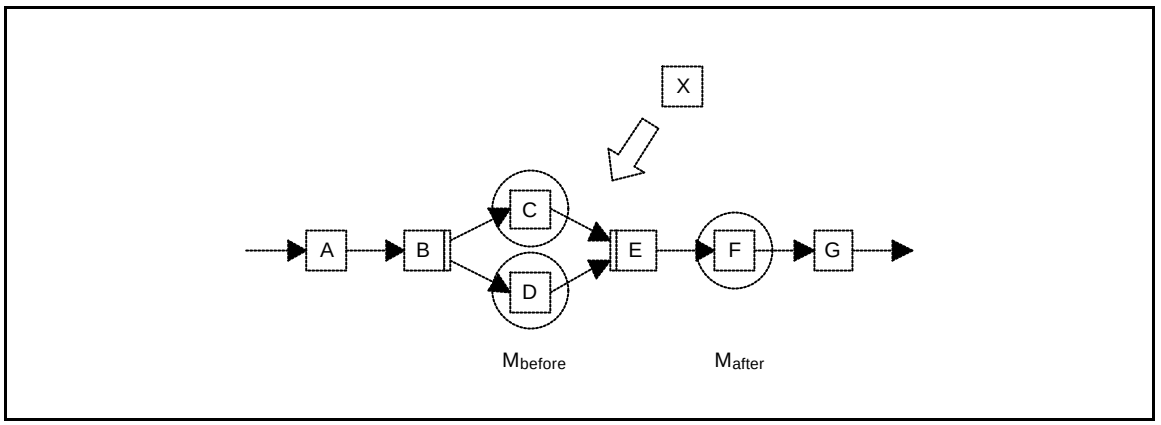


Abbildung 3.8: Dynamisches Einfügen in ADEPT - Festlegen der Position

Um zu garantieren, daß die Einfügeoperation wieder zu einem konsistenten und korrekten Workflow führt, müssen gewisse Vorbedingungen gelten:

- Jeder Knoten aus M_{before} muß im Kontrollfluß vor jedem Knoten aus M_{after} stehen.
- Knoten aus M_{before} dürfen nicht gegenseitig voneinander abhängen. Gleiches gilt für Knoten aus M_{after}.
- Die Blockstruktur innerhalb der Knoten von M_{before} und M_{after} darf nur komplette Schleifenkonstrukte enthalten.

Unabhängig von den strukturellen Vorbedingungen dürfen die Knoten aus M_{after} selbstverständlich weder bereits beendet sein, noch sich aktuell in Ausführung befinden. Ansonsten müßten sie zurückgesetzt werden. Der Zustand der Knoten aus M_{before} ist nebensächlich.

Sind die Vorbedingungen erfüllt, kann der Algorithmus ansetzen, den minimalen geschlossenen Block im Workflow zu finden. Der minimale geschlossene Block enthält alle Knoten von M_{before} und M_{after}, sowie alle dazwischenliegen Knoten. Sollte diese Knotenmenge ein nicht komplettes Schleifenkonstrukt bilden, wird noch eine minimale Menge außerhalb liegender Knoten hinzugenommen. In Abbildung 3.9 ist dies Block B.

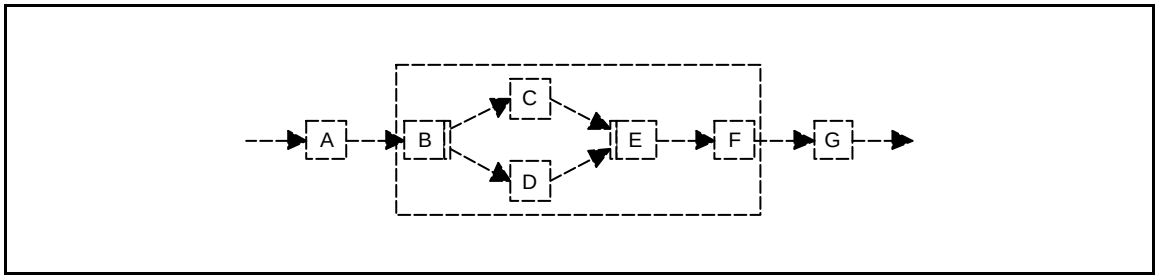


Abbildung 3.9: Dynamisches Einfügen in ADEPT - Ermitteln des minimalen geschlossenen Blocks

Als nächstes fügt der Algorithmus zwei zusätzliche aktivitätenlose Knoten in den Workflow ein. Vor dem vorhin festgelegten minimalen geschlossenen Block eine UND-Verzweigung (n_1) und nach dem Block eine UND-Vereinigung (n_2). Diese Knoten übernehmen entsprechend die ein- und ausgehenden Kontrollkanten. Nun kann der neue Knoten X als paralleler Zweig zum Block eingefügt werden. Als letzter Schritt werden durch das Hinzufügen von Synchronisationskanten (SOFT_SYNC) die Abhängigkeiten zwischen alten und neuen Knoten dargestellt (Abbildung 3.10). Die Synchronisationskanten verlaufen von den Knoten C und D (M_{before}) zum Knoten X, und von diesem wiederum zum Knoten F (M_{after}).

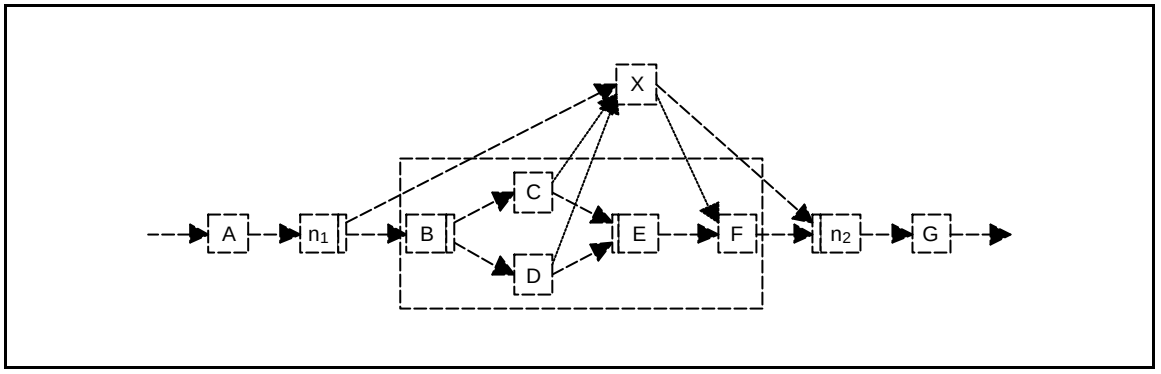


Abbildung 3.10: Dynamisches Einfügen in ADEPT - Kontrollkanten und Synchronisationskanten einfügen

Anschließend daran wird noch versucht durch bestimmte Regeln den neuen Aufbau des Workflows auf die wesentlichen Knoten zu reduzieren, indem die eingefügten aktivitätenlosen Knoten wieder entfernt werden und der Kontrollfluß sinngemäß angepaßt wird (Abbildung 3.11).

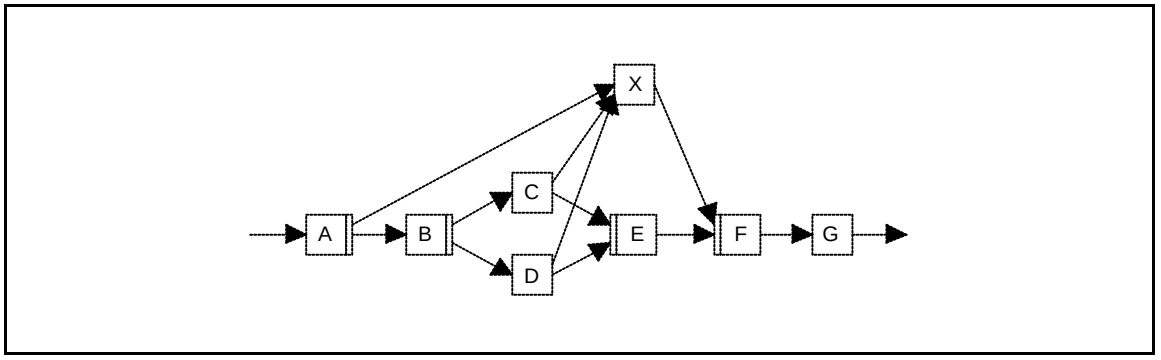


Abbildung 3.11: Dynamisches Einfügen in ADEPT - Reduktion des geänderten Workflows

Die folgende Neubewertung des Workflow-Zustandes beruht auf der Workflow-Ausführung und den Signalen, wie sie mit ADEPT definiert wurden. Sie beinhaltet das Neubewerten aller, bereits vor der Änderung bestandenen Knoten und Kanten, sowie der neu eingefügten. Ob eine neu eingefügte Aktivität direkt aktiviert wird, hängt also wie gehabt, von den Zuständen der im Workflow-Graph davorliegenden Aktivitäten ab. In unserem Beispiel wird die Aktivität X sofort aktiviert, wenn die Aktivitäten A, C und D erfolgreich durchgeführt wurden (Zustand COMPLETED) und damit alle drei in X einlaufenden Kanten auf TRUE_SIGNALED stehen.

Die letztlich noch nötige Anpassung der Datenflußbindungen kann ein komplizierterer Vorgang sein. Die Parameter der geänderten Aktivitäten müssen über Data-Links passend mit den bestehenden Variablen verbunden werden. Eventuell müssen auch noch neue Variablen mit alten Aktivitäten verknüpft werden. Dies muß selbstverständlich unter Beachtung der Regeln der Datenflußmodellierung geschehen.

3.1.6 Beispiel in ADEPT

In seiner Grundform ist ADEPT nicht in der Lage, Produktentwicklungsprozesse mit einer, nicht von Anfang an modellierbaren, flexiblen Anzahl von parallelen Entwicklungsschritten zu unterstützen. Für solche Anwendungen wurde es jedoch um die Methoden von $ADEPT_{flex}$ erweitert. $ADEPT_{flex}$ bietet eine spezielle Unterstützung für dynamische Änderungen an einem Workflow-Schema. Dies betrifft die Operationen des Einfügen und Löschen von Aktivitäten in einem Workflow, der bereits ausgeführt wird. Damit gibt es prinzipiell zwei Möglichkeiten in ADEPT einen Workflow für Produktentwicklungsprozesse mit einer variablen Anzahl paralleler Kontrollflüsse zu modellieren.

Die erste Methode verwendet die Einfüge-Operation. Sie geht von einem sehr kleinen Grund-Schema aus, das nur aus den zwei Aktivitäten ‘Lege Konstruktionsumfang fest’ und ‘Führe Freigabe durch’ besteht (Abbildung 3.12). Die sich normalerweise zwischen diesen Aktivitäten befindende Entwicklung und Prüfung der Bauteile, kann zu Anfang noch nicht festgelegt werden.

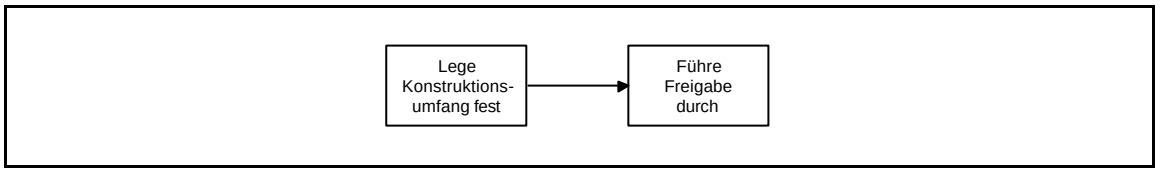


Abbildung 3.12: Workflow-Grund-Schema bei Einfüge-Methode

Nach dem Start dieses Workflows wird die erste Aktivität ausgeführt. Wird dabei festgestellt, daß diverse Bauteile zu entwickeln beziehungsweise zu ändern sind, so müssen die dafür nötigen Aktivitäten dynamisch in das Workflow-Schema eingebaut werden. Nach dem Einfügen eines Einzelteils sieht das Workflow-Schema aus, wie in Abbildung 3.13 dargestellt.

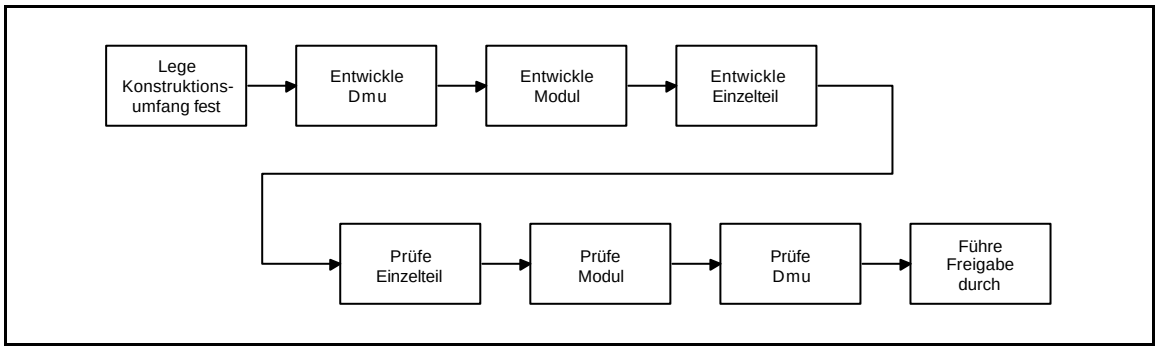


Abbildung 3.13: Workflow-Schema nach dem Einfügen des ersten Einzelteils

Für das Einfügen eines einzigen Einzelteils müssen sechs Aktivitäten eingefügt werden, da die entwickelten Einzelteile nach der vorgegebenen Datenstruktur (siehe Kapitel 1.3) nicht nur auf der Einzelteil-Ebene, sondern auch noch auf der Modul- und der Dmu-Ebene zu prüfen sind. Die zusätzlichen einleitenden Aktivitäten ‘Entwickle Dmu’ und ‘Entwickle Modul’ sind zwar prinzipiell für die Entwicklung eines Einzelteils nicht erforderlich, sie werden aber benötigt, um bei weiteren Einfügungen die symmetrische Blockstrukturierung zu erhalten. Ist die Entwicklung zusätzlicher Bauteile nötig, so vergrößert sich natürlich auch die Anzahl der einzufügenden Aktivitäten. Handelt es sich um ein Bauteil eines bereits eingefügten Moduls oder einer bereits eingefügten Dmu, so sind natürlich entsprechend weniger zusätzliche Einfüge-Operationen

erforderlich, da die Prüfung des Moduls, ebenso wie die Prüfung der Dmu bereits vorhanden ist. Ein Beispiel für das Einfügen zweier weiterer Einzelteile innerhalb eines bestehenden Moduls zeigt Abbildung 3.14.

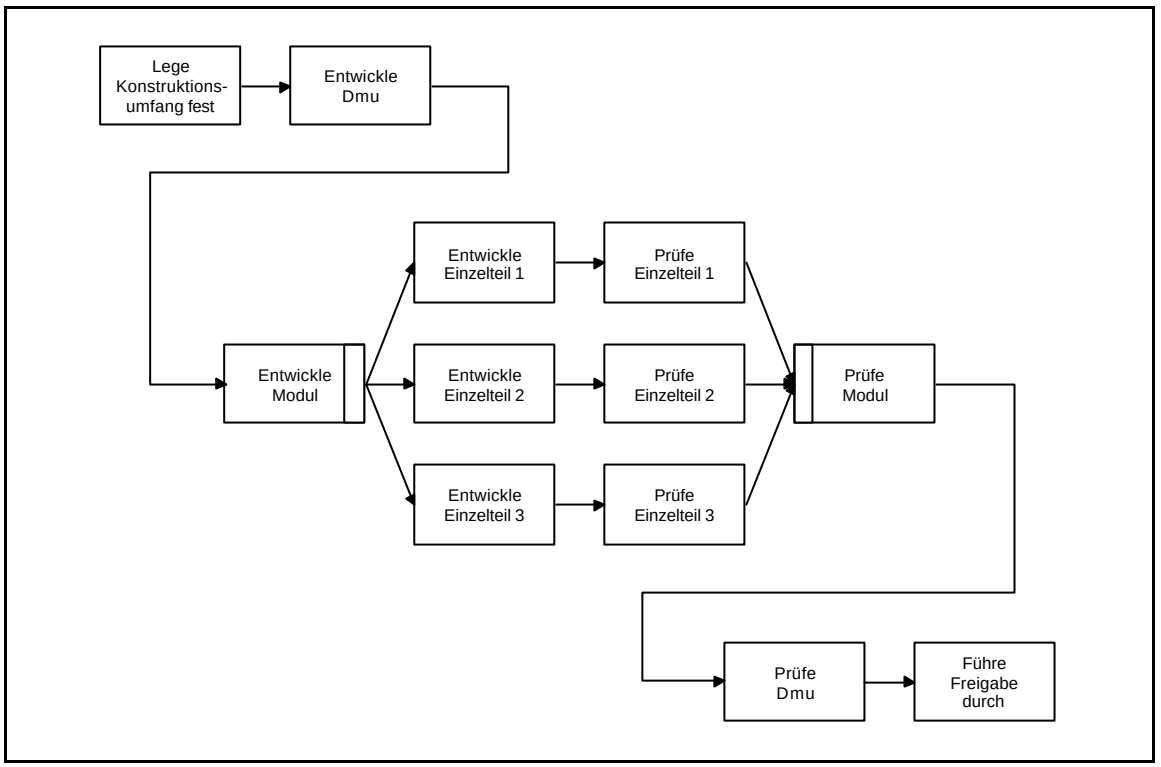


Abbildung 3.14: Workflow-Schema nach dem Einfügen eines weiteren Einzelteils

Die so durch das Einfügen entstandenen parallelen Kontrollflußzweige müssen selbstverständlich alle abgearbeitet werden, weshalb sie sich innerhalb von UND-Verzweigungen befinden müssen. Durch das Einfügen weiterer Einzelteile, und eventuell benötigter Prüfungen auf Modul- und Dmu-Ebene kann das Workflow-Schema sehr schnell sehr groß werden. Es bleibt zwar ständig auf der Basis der symmetrischen Blockstrukturen, jedoch ist der Aufwand für eine Einfüge-Operation natürlich immer vorhanden. Er steigt dabei entsprechend der Anzahl einzufügender Einzelteile in etwa linear. Zusätzlich zu den neuen Aktivitäten müssen auch immer neue Datenvariablen und die zu den Ein- und Ausgabedaten der Aktivitäten passenden Datenflußbeziehungen erstellt werden. Synchronisationskanten werden nicht benötigt, da die Abhängigkeiten anhand der Datenstruktur streng hierarchisch geordnet sind und immer nur im selben Kontrollfluß nachfolgende Aktivitäten auf die Ausgabedaten voriger Aktivitäten angewiesen sind.

Die Ausführung der Aktivitäten geschieht strikt in der modellierten Reihenfolge und immer erst, wenn alle Vorgänger-Aktivitäten erfolgreich beendet wurden. In Abbildung 3.14 beispielsweise, wird die Aktivität ‘Prüfe Modul’ erst ausgeführt, wenn alle drei ‘Prüfe Einzelteil’-Aktivitäten den Status COMPLETED erreicht haben und die ausgehenden Kanten TRUE_SIGNALED melden. Eine vorzeitige Datenweitergabe erfolgt nicht.

Die zweite Methode, einen solchen Produktentwicklungsprozeß zu modellieren, arbeitet mit der Lösch-Operation. Sie basiert im Gegensatz zur ersten Methode auf einem großen Grund-Schema. Dieses Grund-Schema besteht zu Anfang aus dem maximalen Schema, das darstellbar ist (Abbildung 3.15). Dann wird gewissermaßen genau andersherum vorgegangen wie bei der ersten Methode. Nach und nach werden die Aktivitäten, die für die Entwicklung oder Änderung nicht erforderlich sind, aus dem Workflow-Schema entfernt. Dieser Vorgang läuft so lange, bis

genau der Workflow entstanden ist, der für die Bearbeitung ausreicht. Das heißt, bis im Workflow nur noch die Einzelteile entwickelt werden, die im Konstruktionsumfang festgelegt sind.

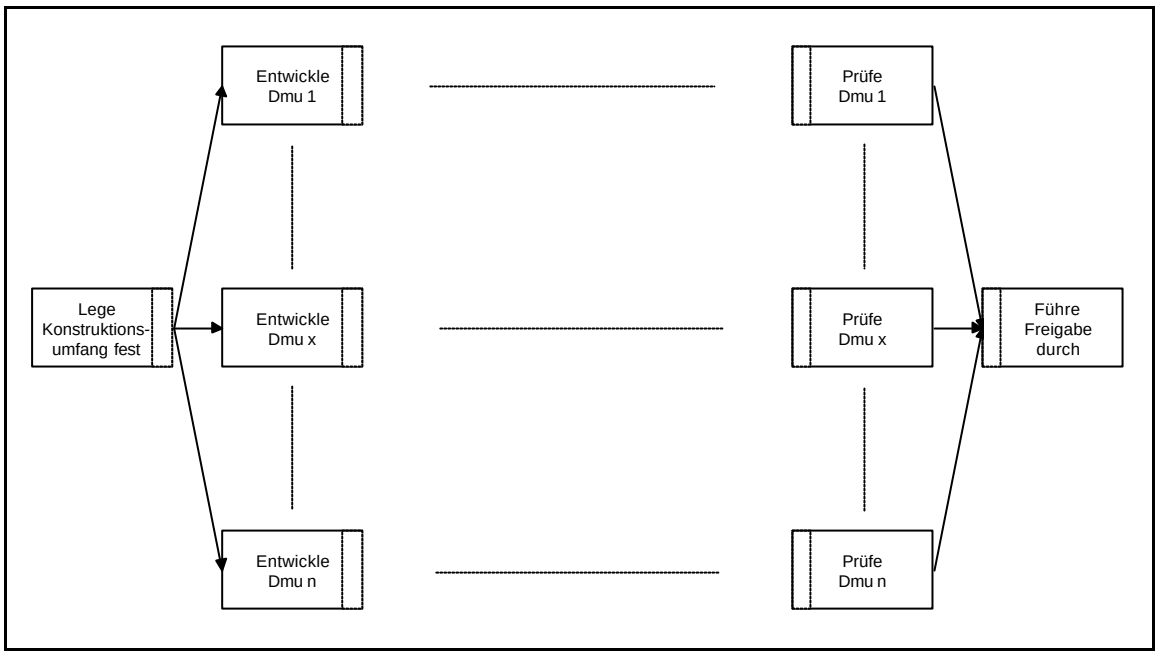


Abbildung 3.15: Workflow-Grund-Schema bei Lösch-Methode

Letztenendes entsteht bei Methode zwei dasselbe Workflow-Schema wie nach Methode eins, nur auf dem umgekehrten Weg. Das größte Problem der Einfüge-Methode dürfte der sicherlich recht hohe Aufwand für die vielen Einfüge-Operationen sein. Dieser Aufwand ließe sich durch die Möglichkeit, mehrere Aktivitäten gleichzeitig einzufügen, verringern. Ein Workflow nach dem oben beschriebenen Schema könnte so beispielsweise aus mehreren vorgefertigten Workflow-Bauteilen zusammengesetzt werden. Konsistenzprobleme können dabei, wie auch beim einfachen Einfügen, durch die eindeutige Datenzuordnung der hierarchischen Datenstruktur und die nicht nötigen Synchonisationskanten kaum auftreten.

Das Löschen von Aktivitäten verursacht sicherlich weniger Aufwand als das Einfügen. Somit hat hier die Lösch-Methode einen Vorteil. Sehr nachteilig wirkt sich aber aus, daß das Workflow-Schema von Grund auf in der Lage sein muß, jede erdenkliche Datenstruktur zu bearbeiten. Ganz egal, wie groß diese ist. Das Workflow-Grund-Schema muß also sehr umfangreich sein. Methode zwei hat damit einen erheblichen Modellierungs- und Speicheraufwand. Auch in dem Fall, daß nur ein einziges Einzelteil zu bearbeiten ist, muß am Anfang die maximal mögliche Anzahl von Einzelteilen unterstützt werden.

3.1.7 Abgrenzung und Bewertung

ADEPT wurde entwickelt, um Workflow-Techniken für den klinischen Bereich anzupassen. Es basiert prinzipiell auf einem Standard-Workflow-Management-Modell, das um diverse Konzepte zur dynamischen Anpassung während der Laufzeit erweitert wurde. Aufbauend auf einem graphenorientierten Modell, bevorzugt ADEPT damit stark vorstrukturierte Prozesse, die

häufigen Wiederholungen unterliegen. Die Methoden, die ADEPT anbietet, um dynamische Änderungen in einem bereits laufenden Workflow durchzuführen, sind für kleine Umgestaltungen sehr gut geeignet und sichern durch eine formale Grundlage die Korrektheit und Konsistenz des Workflows. Damit eignen sie sich gut für Workflows, die geändert werden müssen, deren Abarbeitung aber nicht gestoppt werden kann. Dies ist sicher ein häufig anzutreffender Fall in klinischen Anwendungsbereichen. Aber auch in vielen anderen Gebieten, in denen eine dauernde Überwachung garantiert werden muß, kann dies ein wichtiges Kriterium sein.

Für Produktentwicklungsprozesse ist ADEPT dagegen weniger gut geeignet. Auf der Ebene der Grobplanung, das vormodellierte Grund-Schema betreffend, haben die Modelle von ADEPT und WEP noch einiges gemein. Abgesehen von den unterschiedlichen Aktivitäten-typen, sind sie sich in der Kontrollflußmodellierung, wie auch in der Datenflußmodellierung sehr ähnlich. Beide Modelle basieren im Kontrollfluß auf einer symmetrischen Blockstruktur und im Datenfluß auf globalen Datenvariablen und der Beschreibung der Abhängigkeiten zwischen den Eingabe- und Ausgabedaten der Aktivitäten und den Datenvariablen.

ADEPT bietet dagegen keine direkte Unterstützung für die Feinplanung, das heißt, für die kreative Entwicklungsarbeit innerhalb eines konstruktiven unstrukturierten Teilprozesses, da es eine Aktivität direkt mit der Ausführung eines Werkzeuges gleichsetzt. Somit ist dem Bearbeiter die Arbeitsreihenfolge vorgegeben, da vormodelliert. WEP verbindet verschiedene Werkzeuge innerhalb einer zielorientierten Aktivität ohne einen Ablauf vorzugeben. Dies gewährt dem Bearbeiter mehr Freiheit in seiner Arbeitsweise. Unstrukturierte Teilprozesse können in ADEPT nur durch eine künstlich vorgegebene Standardreihenfolge modelliert werden, von der dann mit den dynamischen Umstrukturierungsmöglichkeiten manuell abgewichen werden kann.

Weiterhin fehlt dem ADEPT-Modell ein einfaches Konstrukt zur Unterstützung variabler Parallelität, wie es das Traversierungsmerkmal in WEP darstellt. Bei Entwicklungsprozessen muß zur Laufzeit entschieden werden, welche Bauteile zu entwickeln oder zu ändern sind. Dies kann nicht zur Modellierungszeit geschehen. Die Methoden, die $ADEPT_{flex}$ hier zur Verfügung stellt, sind für die großen Datenmengen von Entwicklungsprozessen zu aufwendig, da sie mit vielen manuellen Restrukturierungen zur Laufzeit verbunden sind. Wie im Beispiel in Kapitel 3.1.6 zu sehen, müßte mit diesen Methoden mit jeder erneuten Ausführung des entsprechenden Workflows das Workflow-Schema zur Laufzeit anhand des ebenfalls neu festgelegten Konstruktionsumfangs angepaßt werden. Ob dies nun durch das Einfügen der benötigten Aktivitäten oder das Löschen der nicht benötigten Aktivitäten geschieht, ist nicht von Belang. Es führt in jedem Fall zu einem enormen manuellen Zusatzaufwand zur Laufzeit, der mit jeder neuen Workflowausführung erneut anfällt. In WEP dagegen wird die variable Zahl der parallelen Kontrollflußzweige durch das Traversierungs-merkmal automatisch aus der Datenbelegung abgeleitet.

Weitergehende Mechanismen zur Beschleunigung von Entwicklungsprozessen bietet ADEPT nicht an. So gibt es keine Ansätze für Simultaneous Engineering, da eine Folgeaktivität erst dann gestartet wird, wenn die vorige Aktivität beendet wurde. Dies erlaubt auch keine Möglichkeiten der vorzeitigen Datenweitergabe. Desweiteren fehlt ADEPT bislang ein adäquates Zeitmanagement, das in der Produktentwicklung zur Einhaltung von Fristen unumgänglich ist.

3.2 CONCORD

CONCORD steht für **CON**trolling **CO**ope**R**ation in **D**esign environments. Durch diese Benennung wird bereits deutlich, daß bei CONCORD sehr viel Wert auf die Unterstützung und Kooperation von am Projekt Beteiligter gelegt wird. In diesem Zusammenhang wird beim CONCORD-Modell lieber von **Designflow-Management** als von Workflow-Management gesprochen. Bei Designflow-Management liegt dabei der Schwerpunkt auf einer reinen Assistenzfunktion, d.h. die Bearbeiter sollen durch ihren Entwurfsauftrag geführt werden ohne in ihrer Kreativität eingeschränkt zu werden. Die Unterstützung hierzu umfaßt nicht nur vollkommen geregelte (vorplanbare), sondern auch ungeregelte (nicht im einzelnen vorhersehbare) Abläufe, wobei insbesondere letztere auch die Unterstützung einer weitgehend freien Kooperation unter Konsistenzhaltung der verwendeten Daten (Entwurfsdaten) erfordern. Workflow-Management dagegen wird als eher strikte Vorgehensweise angesehen mit spezifischen Systemlösungen wie zum Beispiel Produktions-Workflows.

Was ist nun genau unter der Assistenzfunktion des Designflow-Managements (vgl. [RiMi97]) zu verstehen? Betrachten wir einen voll vorgeplanten Ablauf, so braucht dem Bearbeiter im Treffen von Ablaufentscheidungen nicht assistiert zu werden, da alle Ablaufentscheidungen durch die Workflow-Spezifikation vorweggenommen wurden. Im Falle des anderen Extrems, eines völlig ungeregelten Ablaufs, kann nicht assistiert werden, da dem System keine Grundlagen für die Vorbereitung von Ablaufentscheidungen vorab bekannt gemacht werden können. Liegt jedoch ein teilweise geregelter und teilweise ungeregelter Ablauf vor, so sind dynamisch Ablaufentscheidungen durch den Benutzer zu treffen, wobei das System assistieren kann. Die Assistenz beinhaltet Hilfen im Treffen von Entscheidungen, indem dem Benutzer Lösungsmöglichkeiten angeboten werden, und garantiert, daß die dynamischen Entscheidungen mit den vorgegebenen Spezifikationen konform sind. Darunter fällt die Korrektheit der Abläufe und die Konsistenz der Entwurfsdaten.

Für die Assistenzfunktion des Designflow-Management-Systems sind besonders die drei folgenden Aspekte von herausragender Wichtigkeit:

- Unterstützung von effizienten Zugriffsmöglichkeiten auf Datenbereiche (von lokalen, auftragsbezogenen Daten bis hin zu gemeinsamen Daten)
- Erfassung von sowohl konkreter wie auch abstrakter Spezifikationen der Vorplanung (von vollständiger Vorplanung bis hin zu absolut dynamischen Entscheidungen)
- Bereitstellung von flexiblen Kooperationsmechanismen (von isolierter Auftragsbearbeitung bis hin zu freier Kooperation)

Dennoch können viele Mechanismen aus dem Workflow-Management auch im Designflow-Management sinnvoll eingesetzt werden. Dazu zählen insbesondere grundlegende Kooperationsdienste, Techniken der Datenbankanbindung und Transaktionsunterstützung.

3.2.1 Modell

Der CONCORD-Ansatz (vgl. [RMH+94]) erhebt den Anspruch die Komplexität eines Entwurfsvorgangs beherrschbar und die Abläufe soweit wie möglich kontrollierbar zu machen. Das verwendete Modell versucht dazu die natürliche Abbildung von Entwurfsabläufen durch die

Unterteilung in drei **Abstraktionsebenen** zu erreichen. Die drei Ebenen werden in Abbildung 3.16 dargestellt.

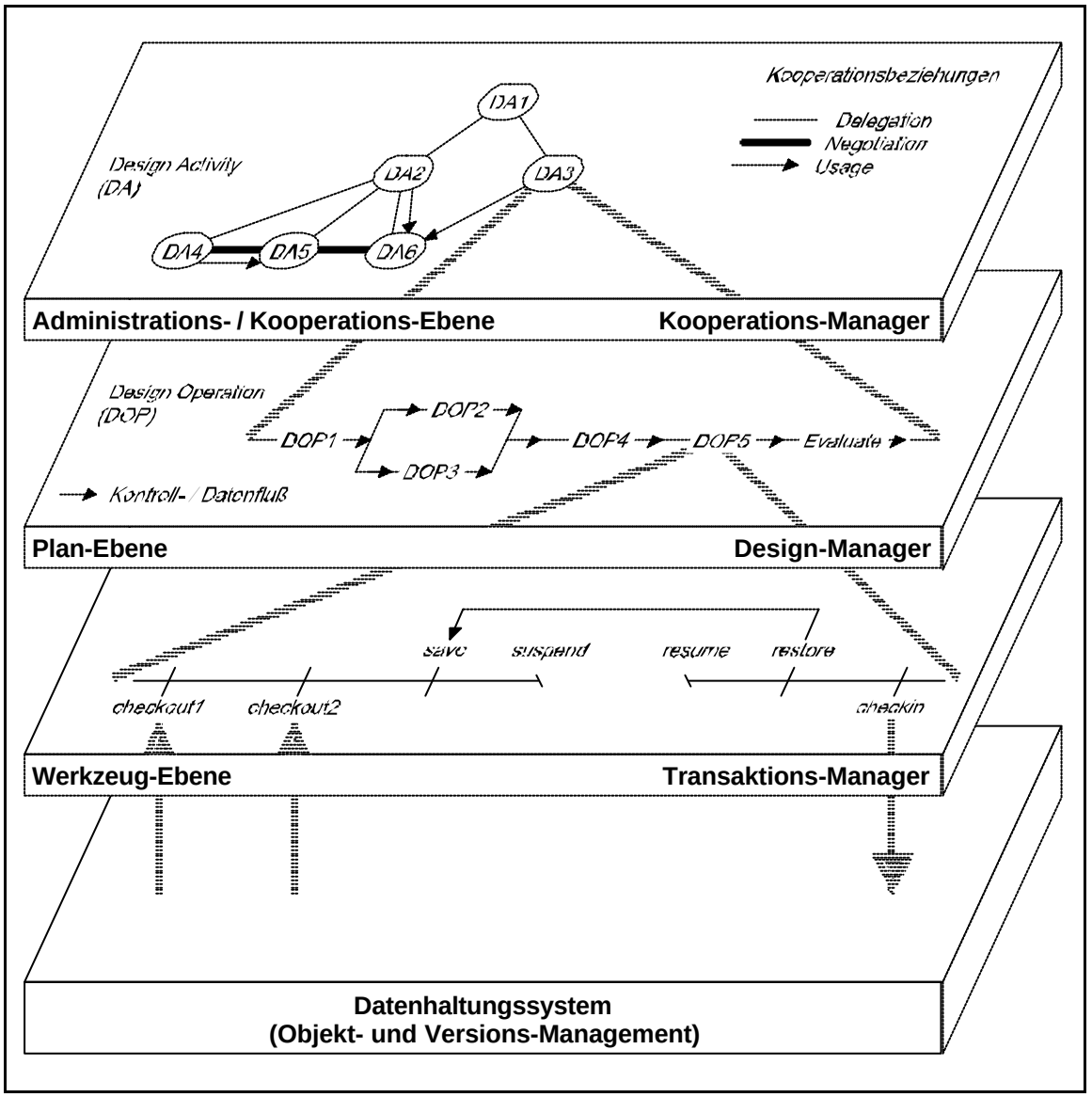


Abbildung 3.16: Abstraktionsebenen des CONCORD-Modells

Die Ebenen des Modells sind von oben nach unten (Top-Down) zu betrachten. Die oberste Ebene ist die **Administrations-/Kooperations-Ebene**. Sie soll den kreativen und administrativen Teil der Entwurfsarbeit unterstützen. Darunter folgt die **Plan-Ebene**, in ihr steht die Anordnung bestimmter atomarer Entwurfsschritte zwecks Erreichung des Entwurfszieles im Vordergrund. Die dritte Ebene ist die **Werkzeug-Ebene**. Sie versucht die Ausführung eines Werkzeugs und Erstellung neuer Versionen der Daten als Transaktion darzustellen.

Unter den drei Abstraktionsebenen befindet sich noch ein **Datenhaltungssystem**, das für die Speicherung der Daten und das Versionsmanagement verantwortlich zeichnet.

3.2.2 Administrations-/Kooperations-Ebene

Die höchste Abstraktionsebene ist die Administrations-/Kooperations-Ebene (AC level, administration/cooperation level). Der Schwerpunkt liegt hier auf der Beschreibung und Zuordnung von Ablaufeinheiten (z.B. Arbeitsaufträgen), sowie auf der Kontrolle der Kooperation zwischen den Ablaufeinheiten.

Das zentrale Konzept dieser Ebene ist die **Design Activity (DA)**. Eine Design Activity ist eine Ablaufeinheit, die alle notwendigen Schritte der Bearbeitung eines Auftrages beziehungsweise Teilauftrages umfaßt. Während des Entwurfsprozesses kann eine Hierarchie von DAs dynamisch aufgebaut werden, die eine Hierarchie von nebenläufigen (Teil-) Aufträgen darstellt (Abbildung 3.17). Idealerweise kann jeder DA jeweils ein Bearbeiter zugeordnet werden (Concurrent Engineering).

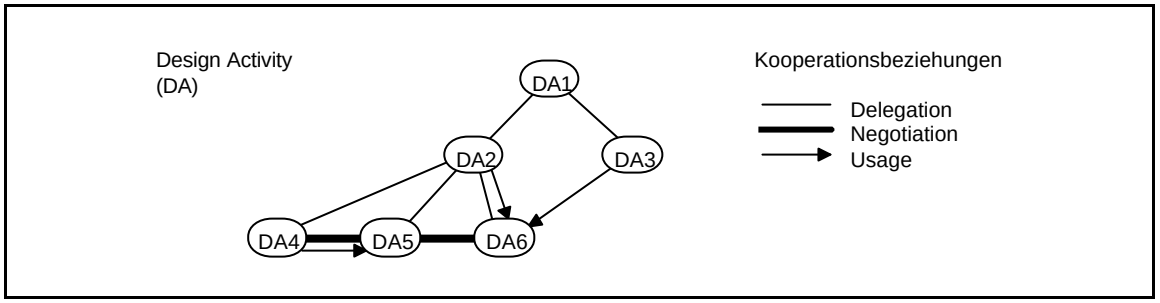


Abbildung 3.17: Administrations- / Kooperations-Ebene - Hierarchie von Design Activities

Der Entwurfsauftrag einer jeden DA ist in Form einer expliziten Spezifikation erfaßt. Ferner werden verschiedenartige Kooperationsbeziehungen explizit modelliert. Die Beziehungstypen heißen **Delegation**, **Negotiation** und **Usage**.

Eine Auftragsbeziehung (Beziehungstyp **Delegation**) beschreibt den Designflow. Sie entsteht implizit bei der Auslagerung eines Teil-Auftrages durch Erzeugung einer untergeordneten DA. Die untergeordnete DA wird auch Sub-DA genannt, während die übergeordnete DA als Super-DA bezeichnet wird. Dies könnte in unserem obigen Beispiel so aussehen, daß DA1 (Computer) Entwurfsaufgaben weiterdelegiert, so zum Beispiel an DA2 (Systemkomponente) und an DA3 (Gehäuse). Delegationsbeziehungen spannen also den Hierarchie-Baum der Design Activities auf.

Kooperationsbeziehungen (Beziehungstyp **Usage**) dienen dem kontrollierten Austausch von Entwurfsdaten. Diese können auch vorläufig sein. So benötigt in unserem Beispiel DA3 (Gehäuse) als vorläufige Daten die Größe von DA6 (Systemplatine).

Der dritte Beziehungstyp, die Verhandlungsbeziehung (**Negotiation**) ermöglicht die Abstimmung der Entwurfsspezifikationen kooperierender Design Activities. An unserem Beispiel zwischen DA6 (Systemplatine) und DA5 (Grafikkarte) könnte dies die Abstimmung über die benötigte Schnittstelle sein.

Eine Design Activity wird mittels eines **Beschreibungsvektors** initialisiert. Dieser Vektor enthält vier Parameter: <DOT, SPEC, Designer, DC>.

DOT steht für Entwurfsobjekttyp (design object type) und beinhaltet die Typinformation der für eine DA relevanten Entwurfsdaten, wie zum Beispiel die Startzustände der Entwurfsobjekte. Ein DOT kann dabei ein einfacher Typ oder ein strukturierter Typ sein. Ausprägungen einfacher Typen sind Versionen, Ausprägungen strukturierter Typen sind Konfigurationen. Genaueres

dazu findet sich in [RMHN95]. Für unsere abstrakte Betrachtungsweise reicht es, diese Versionen und Konfigurationen verallgemeinert nur als Entwurfsobjektzustand (DOS, design object state) zu betrachten. Alle Zustände der Entwurfsobjekte die während des Ablaufs der DA erstellt werden sind in einem Abhängigkeitsgraph organisiert. Eine DA kann, ohne besondere Authorisierung, nur auf ihren eigenen lokalen Abhängigkeitsgraphen zugreifen, alle anderen sind für sie nicht sichtbar.

Alle Aktivitäten innerhalb einer DA zielen letztendlich darauf ab, einen bestimmten Entwurfsauftrag oder Teilauftrag zu erfüllen. Dieser Auftrag wird beschrieben durch eine explizite Entwurfszielspezifikation (design specification), die sich im Beschreibungsvektor hinter dem zweiten Parameter **SPEC** verbirgt. Die Spezifikation besteht aus einer Reihe von geforderten Eigenschaften (Features). Im einfachsten Fall handelt es sich dabei um eine Attribut-Wertebereichs-Bedingung, es kann sich aber auch beispielsweise um eine Forderung handeln, daß ein bestimmtes Testwerkzeug erfolgreich passiert werden muß. Im Hinblick auf die Zielspezifikation können den DOSs einer DA verschiedene Qualitätsstufen zugeordnet werden. Dies ist wichtig, um während einer DA herauszufinden, welche Schritte zum Erreichen der Zielspezifikation noch durchgeführt werden müssen. Vereinfacht spricht man von einem vorläufigen DOS, solange noch nicht alle Eigenschaften erfüllt sind und von einem Ergebnis-DOS, sobald dessen Qualitätsstufe der Zielspezifikation entspricht.

Der dritte Parameter **Designer** ordnet der Design Activity den verantwortlichen Bearbeiter zu. Dieser ist verantwortlich für die Aktionen die innerhalb der DA ausgeführt werden.

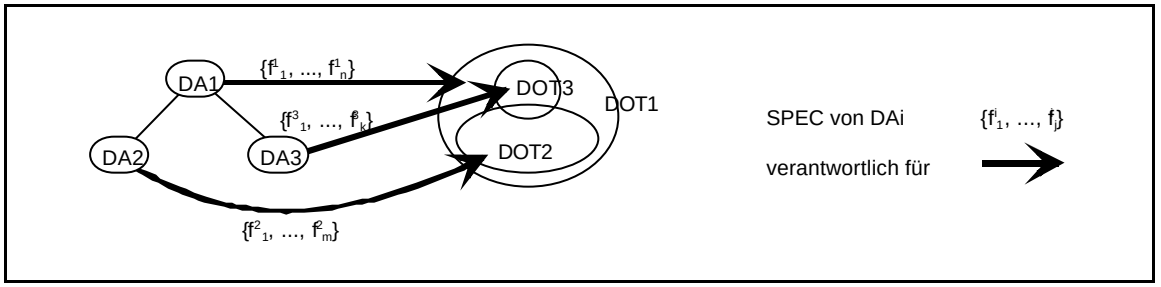


Abbildung 3.18: Delegation von Sub-DAs

Der vierte Parameter **DC** (design control) enthält Ablaufsteuerungsinformationen, wie die Workflow-Struktur innerhalb der DA oder die Auslagerung von Teilaufträgen durch Erzeugung von Sub-DAs. Die Erfüllung der Teilaufträge ist natürlich Voraussetzung für die erfolgreiche Bearbeitung des Auftrages der Super-DA. Die Delegation wird sich in aller Regel an der DOT-Struktur der Super-DA orientieren, dabei sind wie in Abbildung 3.18 die DOTs der Sub-DAs ein Teil des DOT der Super-DA. Die Delegation kann sich dabei auf eine komplette Aufteilung der Entwurfsaufgabe in mehrere Teilaufträge beziehen oder auch nur auf die Abspaltung einiger kleinerer Aufgaben vom Hauptauftrag. Auf jeden Fall muß die Super-DA dafür Sorge tragen, daß die Ergebnisse der Sub-DAs wieder korrekt zusammengeführt beziehungsweise integriert werden.

Neben der auftragsbezogenen Kommunikation entlang von Delegationsbeziehungen werden durch den CONCORD-Ansatz auf der Administrations-/Kooperations-Ebene zwei weitere Arten der Kooperation unterstützt. Bei beiden Beziehungen wird vorausgesetzt, daß den beteiligten DAs (bzw. dem zugeordneten Bearbeiter) die Zielsetzungen der anderen DAs bekannt sind.

Verhandlungsbeziehungen (**Negotiation**) dienen dem Verhandeln von Zielspezifikationen zwischen Sub-DAs. Der Gegenstand dieser Kooperation ist also eine Entwurfszielspezifikation. Eine DA kann einer anderen DA verfeinerte oder neue Eigenschaften zur Aufnahme in deren

Spezifikation vorschlagen (Propose), um zum Beispiel das nachfolgende Aussehen eines Entwurfsobjekts für alle Beteiligten akzeptabel zu machen. Die angesprochene DA kann diese Vorschläge annehmen oder verwerfen (Agree/Disagree). Können sich zwei verhandelnde DAs nicht einigen, wird die Super-DA informiert, die dann diesen Konflikt auflösen muß.

Kooperationsbeziehungen (**Usage**) dienen dem kontrollierten Austausch vorläufiger Entwurfsdaten zwischen verschiedenen DAs. Aufgrund der Relevanz der Entwurfsqualität sind die Einheiten eines solchen Austausches vom Granulat DOS (design object state). Fordert eine DA von einer anderen einen DOS an (Require), so wird, sofern dieser die geforderte Entwurfsqualität aufweist, eine entsprechende Beziehung aufgebaut. Die Mindestqualität wird durch eine Eigenschaftsmenge spezifiziert. Falls nun ein DOS mit der geforderten Qualität existiert, kann er durch die angesprochene DA zur Nutzung freigegeben werden (Propagate). Die anfordernde DA kann den DOS in die eigenen Berechnungen einbeziehen, muß sich aber darüber im Klaren sein, daß die Daten nur vorläufig sind. Es ist eine Aufgabe des Systems, die entstehenden Abhängigkeiten zu warten und bei Invalidierung eines bereitgestellten DOS entsprechende Maßnahmen, wie die Benachrichtigung der anfordernden DA einzuleiten.

Die Einhaltung der durch diese Beziehungen vorgegebenen Beschränkungen und Semantiken unterliegen der Kontrolle einer zentralen Systemkomponente, dem **Kooperations-Manager**. Der Kooperations-Manager erzwingt, daß eine Kooperation zwischen Design Activities nur auf bereits eingerichteten Kooperationsbeziehungen stattfindet und überprüft, daß jede DA die Integritätsbedingungen der Kooperationsbeziehung einhält. Um diese Aufgabe zu erfüllen, ist die Vorhaltung von Zustandsinformationen für jede DA im Hierarchiebaum unumgänglich. Diese Zustandsinformationen beinhalten den Beschreibungsvektor und Datenbereich einer DA ebenso, wie ihre bestehenden Kooperationsbeziehungen. Im folgenden wird ein vereinfachter Zustands- und Übergangsgraph (siehe Abbildung 3.19) beschrieben, der für jede Design Activity auf der Administrations-/Kooperations-Ebene gepflegt wird.

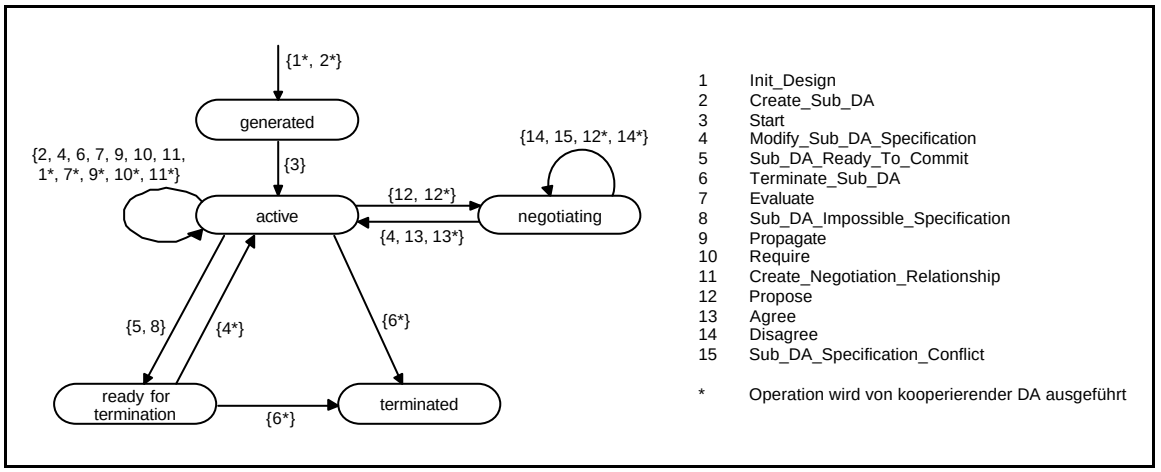


Abbildung 3.19: Vereinfachter Zustands- und Übergangsgraph für eine Design Activity

Der Zustand *generated* wird einer DA zugewiesen, sobald sie mittels Beschreibungsvektor initiiert wird, ihre Arbeit aber noch nicht begonnen hat. Führt die DA ihre Arbeit aus, gelangt sie in den Zustand *active*. Der Zustand *negotiating* wird zugewiesen wenn die DA oder eine andere DA eine neue Objektversion vorschlägt. Die Arbeitsausführung wird dabei unterbrochen. Hat man sich auf eine neue Objektversion geeinigt, geht die DA wieder in den Zustand *active* über und die Ausführung wird fortgesetzt. Nachdem eine DA einen Ergebnis-DOS erzeugt hat, sollte sie dennoch erst beendet werden, wenn die Super-DA das Ergebnis akzeptiert. Dies wird durch den Zustand *ready for termination* erreicht. Der Zustand wird ebenfalls zugewiesen, wenn die DA ihrer Super-DA mitteilt, daß sie nicht in der Lage ist einen Ergebnis-DOS

entsprechend der aktuellen Design-Spezifikationen zu erstellen. Der Zustand *terminated* zeigt schließlich an, daß die DA durch ihre Super-DA beendet wurde.

3.2.3 Plan-Ebene

Die Konzepte der zweiten Abstraktionsebene, der Plan-Ebene (DC level, design control level), werden deutlich, wenn man ins Innere einer Design Activity blickt. Das heißt, sie beschreiben die innere Struktur einer Design Activity. Hier steht das Entwurfsziel einer DA und die Anordnung atomarer Entwurfsschritte zur Erreichung des Ziels im Vordergrund. Die Plan-Ebene betrachtet die kontrollierte Ausführung von Werkzeugen gemäß einer vorgegebenen Methodik und bietet Konzepte zur Spezifikation des Workflows und zur Abarbeitung der die Workflow-Spezifikationen beinhaltenden Skripte. Abbildung 3.20 zeigt grafisch einen Ausführungsplan (Skript) für eine DA. Ein solches Skript besteht aus der Spezifikation von Kontroll- und Datenfluß zwischen einzelnen Werkzeuganwendungen. Es beschreibt also direkt den Workflow.

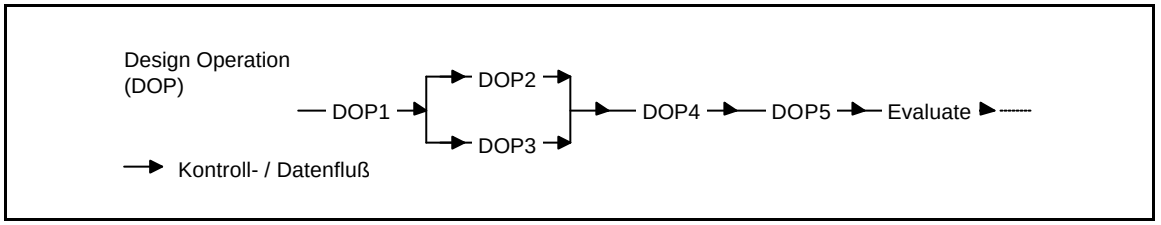


Abbildung 3.20: Workflow einer Design Activity

Der Workflow wird der Design Activity über den vierten Parameter (DC) des Beschreibungsvektors zugewiesen. Das auf dieser Ebene zentrale Konzept ist die **Design Operation (DOP)**. Sie ist der atomare Entwurfsschritt innerhalb der Design Activity, also die Ablaufeinheit zur Ausführung eines Werkzeuges.

Die DOPs werden, entsprechend der vorgegebenen Design-Strategie, in einer spezifischen Reihenfolge ausgeführt. Jeder DOP liest (verschiedene) DOSs aus dem Bereich seiner übergeordneten DA und entspricht einer stückweisen Weiterentwicklung des DOS. Der DOP beendet seine Arbeit mit einem neuen DOS, einer neuen Version eines Entwurfsobjekts, die jedoch ein vorläufiges DOS sein kann. Ein Ergebnis-DOS wird erst am Ende des Workflows erwartet und entspricht dann dem Ergebnis-DOS der übergeordneten Design Activity. Die geforderte Qualität der Entwurfsobjekte wird durch eine spezielle Operation (Evaluate) am Ende des Workflows sichergestellt.

Um die Reihenfolge der Ausführung der DOPs festzulegen ist ein Ausführungsplan (Skript) notwendig. Ein solches Skript enthält Sequenzen, Verzweigungen für nebenläufige Ausführung und Schleifen. Es läßt dem Entwickler deshalb meistens nur geringe Freiheiten. Aus diesem Grunde sind weitere Beschreibungsmöglichkeiten nötig.
Die erste Möglichkeit sind die sogenannten Beschränkungen (constraints). Sie beschränken die Ausführung eines DOP nicht anhand eines Kontrollflusses, sondern durch die Festlegung von Bedingungen. So kann eine DOP beispielsweise nur dann ausgeführt werden, wenn eine andere DOP mit einem festgelegten Ergebnis erfolgreich beendet wurde. Oder eine bestimmte DOP folgt immer einer anderen DOP eines festgelegten Typs.

Die zweite Möglichkeit sind die Ereignis-Bedingung-Aktion-Regeln (event, condition, action rules). Kooperations-Beziehungen zwischen DAs führen zu asynchron auftretenden Ereignissen innerhalb einer DA (Propose, Require) und erwarten eine Antwort oder Reaktion (Agree/Disagree, Propagate) der DA. Solche Konzepte sind am einfachsten durch eine Ausnahmebehandlung durch das Abfragen von Ereignissen zu realisieren.

Die bisherigen Diskussionen hatten generell nur mit der Beschreibung des Kontrollflusses zu tun. Der Datenfluß wird in CONCORD im großen und ganzen nicht beschrieben, da eine Kontrollflußkante eine Datenflußkante impliziert und es sich bei den Daten die sich zwischen DOPs bewegen eigentlich nur um eine DOS-Identifikation und einige Status-Informationen handelt.

Die Korrektheit der Ausführung eines Werkzeuges wird hier durch die Systemkomponente Design-Manager garantiert. Der Design-Manager ist weiterhin für die ebenenspezifische und isolierte Fehlerbehandlung zuständig.

3.2.4 Werkzeug-Ebene

Auf der untersten der drei Abstraktionsebenen, der Werkzeug-Ebene (TE level, tool execution level), werden einzelne DOPs betrachtet. Die Einführung des Begriffes der Design Operation abstrahiert dabei etwas von der einzelnen Werkzeug-Ausführung. Aus Sicht der unter den Abstraktionsebenen liegenden Datenhaltungskomponente (Datenbanksystem) ist eine DOP eine klassische Transaktion mit den bekannten ACID-Eigenschaften (Atomizität, Konsistenz, Isolation, Dauerhaftigkeit). Auch wenn eine DOP, von der Plan-Ebene aus betrachtet, wie eine atomare Operation aussieht, so hat sie doch eine interne Struktur die langlebige Transaktionen unterstützt. Diese interne Struktur wird durch Save/Restore und Suspend/Resume Primitiven eingebracht (siehe Abbildung 3.21).

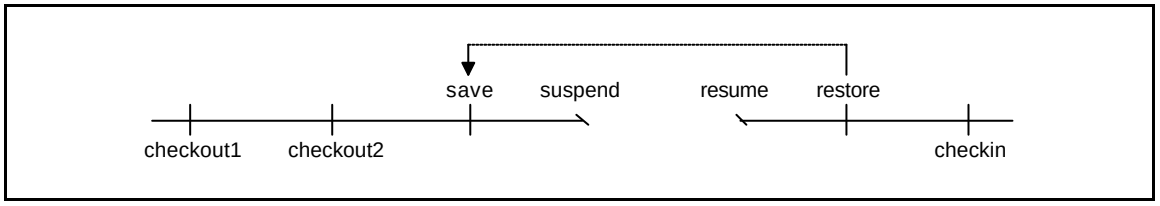


Abbildung 3.21: Ausführung einer Design Operation

Eine Design Operation verarbeitet Objektversionen in drei Schritten. Im ersten Schritt werden Eingabe-Versionen aus der zentralen Datenbank in einen anwendungsnahen Puffer ausgelesen (Checkout). Der zweite Schritt besteht in der Verarbeitung der eingelesenen Daten durch das Werkzeug. Im dritten Schritt wird die geänderte Version beziehungsweise die Änderungsinformation in die Datenbank propagiert (Checkin).

Sicherungspunkte (Savepoints) erlauben dem Entwickler Änderungen und neue Objekte die während seiner Arbeit entstanden sind in die Datenbank auszulagern. Als Konsequenz daraus kann der Entwickler Zwischenzustände sichern (Save). Möchte er später zu einem solchermaßen gesicherten Zustand zurückkehren, so kann er den gesicherten Zustand auswählen und wiederherstellen (Restore). Diese Art der Speicherung und Wiederherstellung kann man sehr einfach als Benutzer-initiiertes Rollback einsetzen.

Um eine DOP für eine längere Zeit auszusetzen, das heißt, pausieren zu lassen, kann man sie unterbrechen (Suspend) und nach beliebiger Zeit wieder fortsetzen (Resume). Der Status der DOP sowie die verwendeten DOSs müssen beim Fortsetzen unverändert, also auf dem gleichen Stand wie beim Unterbrechen, sein.

Die Verwaltung der konsistenten und persistenten Objektversionen wird erneut durch eine Systemkomponente gewährleistet, durch den Transaktions-Manager. Er ist desweiteren zuständig für die isolierte und zurücknehmbare Ausführung der DOPs, die auch auf dieser Ebene für eine ebenenspezifische und isolierte Fehlerbehandlung benötigt wird. Der Transaktions-Manager benutzt außerdem die vom Datenbanksystem zur Verfügung gestellten Operationen und Mechanismen.

3.2.5 Systemarchitektur

Nach der Beschreibung der Konzepte der Abstraktionsebenen, soll hier noch etwas auf die Systemarchitektur von CONCORD und das Zusammenspiel der ebenen-spezifischen Manager eingegangen werden.

Design-Aufgaben, wie sie von CONCORD wahrgenommen werden, laufen meist auf einem verteilten System, im Normalfall auf einem Workstation-Netz. Das verteilte Datenhaltungssystem und seine DBMS-Komponente sind eine typische Serverkomponente und können auf einem einzelnen Serverrechner wie auch in einem verteilten Rechnerverbund arbeiten. Der charakteristische Design-Prozeß eines Bearbeiters ist dagegen eine typische Client-Anwendung für eine einzelne Workstation.

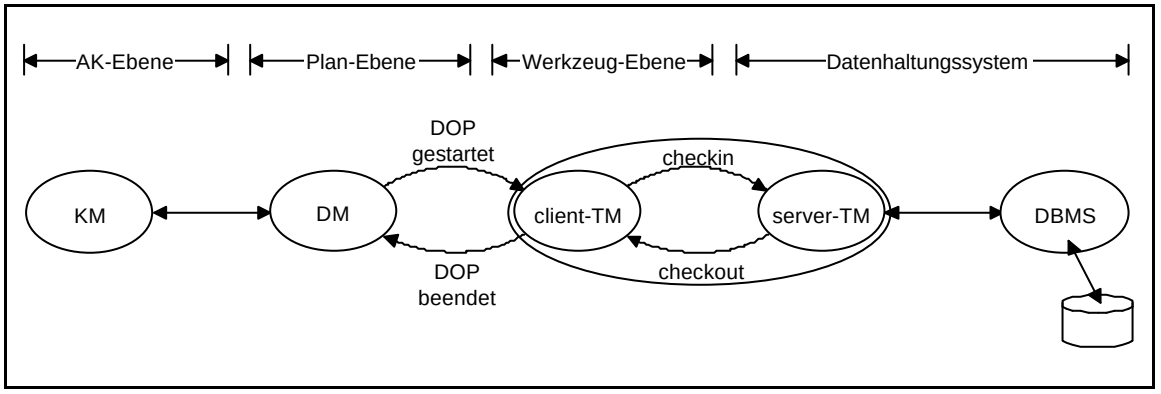


Abbildung 3.22: Zusammenarbeit der ebenen-spezifischen Manager

Da nun in CONCORD eine Design Activity die Arbeit eines einzelnen Bearbeiters darstellt, nehmen wir an, daß sie auf einer einzelnen Workstation läuft. Als Konsequenz daraus kann man erwarten, daß auch alle Operationen (DOPs) die innerhalb einer DA ausgeführt werden, auf dieser Workstation laufen. Gründe dafür sind zum Beispiel das Festlegen von Eingabe-Parametern durch den Bearbeiter und eine notwendige Interaktion zwischen Bearbeiter und Werkzeugen während der Workflow-Ausführung.

Die Assoziation einer DA mit einer Workstation hat direkte Implikationen auf die Zuordnung der ebenen-spezifischen Manager zu ihrer Laufzeit- und Hardware-Umgebung. Der Kooperations-Manager (KM) kümmert sich um die Kontrolle des gesamten Hierarchie-Baums der Design-

Activities und um die Kooperationen dazwischen. Da die DAs über eine große Gruppe von Workstations verteilt sein können, ist es aufgrund der komplexen Beziehungen zwischen einem solchen verteilten System sinnvoll, den Kooperations-Manager als zentrale Komponente serverseitig zu installieren. Dadurch kann er auch relativ einfach das DBMS als Informationsspeicher nutzen. Der Design-Manager (DM), der den DA-internen Workflow und die Abarbeitung der Ausführungsskripte übernimmt, ist auf der Workstation-Seite lokalisiert. Beim Transaktions-Manager (TM) kann dies nicht so einfach festgelegt werden. Er ist zuständig für den verteilten Zugriff aller Bearbeiter und der verwendeten Werkzeuge auf den serverseitigen Datenspeicher und behandelt gleichzeitig die Ausführung der DOPs auf der Client-Seite. Aus diesen Gründen wird er in zwei Komponenten zerlegt. Der Server-TM auf der Server-Seite verwaltet die Checkin- und Checkout-Operationen und kontrolliert den gleichzeitigen Zugriff auf die Versionen der Entwurfsobjekte. Dies geschieht in enger Zusammenarbeit mit dem Datenhaltungssystem. Auf der anderen Seite residiert der Client-TM auf den Workstations (Client-Seite) und ist für die Abarbeitung der internen Struktur der DOPs zuständig.

3.2.6 Beispiel in CONCORD

Unser Beispiel eines Produktentwicklungsprozesses in CONCORD kann folgendermaßen dargestellt werden. Zuerst existiert nur eine Design Activity, sie stellt die gesamte Konstruktion dar. Ihr Beschreibungsvektor enthält in Parameter 1 (DOT) die Beschreibung und den Startzustand des zu erstellenden Kem-Objekts. Parameter 2 (SPEC) beschreibt die Zielspezifikation, das heißt, wie das Kem-Objekt aussehen muß, um die Design Activity als fertiggestellt zu verlassen. Parameter 3 (Designer) legt den oder die Bearbeiter fest. In unserem Beispiel kann nicht, wie idealerweise, der DA nur genau ein Bearbeiter zugewiesen werden. Parameter 4 (DC) enthält die Ablaufsteuerungsinformationen für die DA, wie in diesem Fall die Auslagerung von Teilprozessen.

Der Parameter DC legt somit fest, wie sich in der Entwurfsphase die Hierarchie der Design Activities entwickelt. Er regelt, wie eine DA Teilaufträge an Sub-DAs delegiert. Dabei orientiert sich die Delegation in aller Regel an der DOT-Struktur der Super-DA. Wenn wir jetzt also die Delegation von Sub-DAs anhand des strukturierten Kem-Objekts vollziehen, so müßte eine DA-Hierarchie wie in Abbildung 3.23 entstehen.

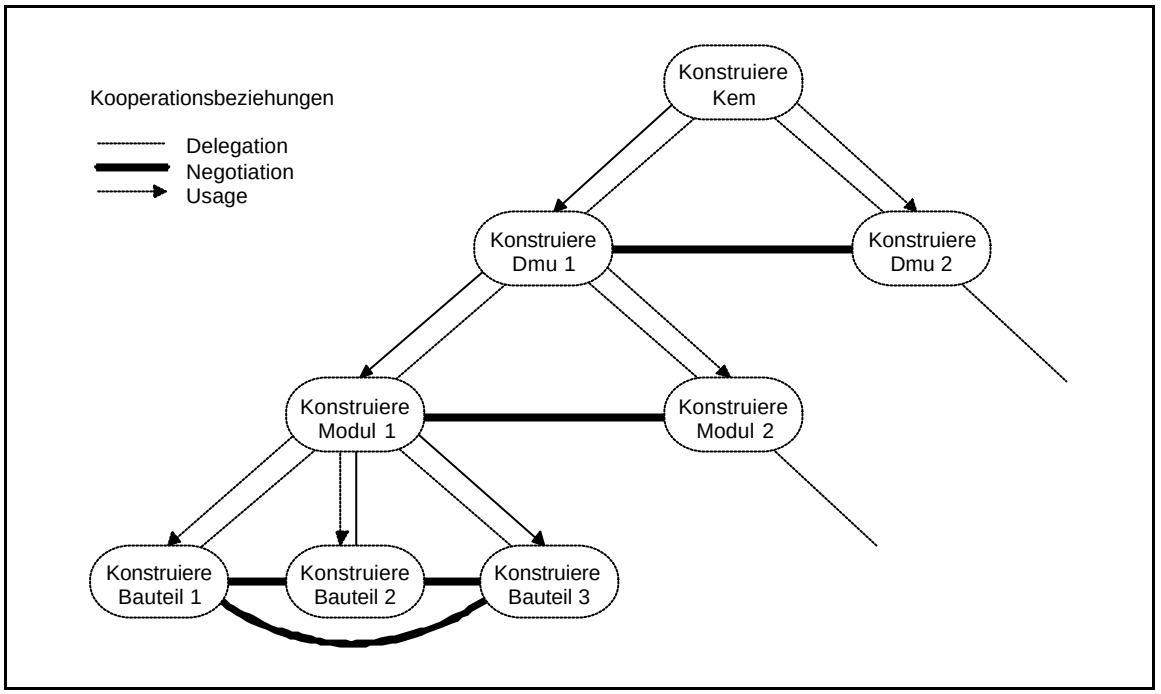


Abbildung 3.23: DA-Hierarchie anhand der Struktur eines Kem-Objekts

Diese DA-Hierarchie teilt also die DA ‘Konstruiere Kem’ in eine oder mehrere DAs ‘Konstruiere Dmu’. Diese werden in die in der Dmu enthaltenen Module und weiter in die in einem Modul enthaltenen einzelnen Bauteile gesplittet. Dargestellt wird dies durch den Beziehungstyp Delegation. Die Beziehungen Negotiation, die jeweils zwischen den DAs einer Ebene bestehen, drücken die Abstimmung aus, die zwischen den jeweiligen Bearbeitern erfolgen muß. Diese Abstimmung kann zum Beispiel sicherstellen, daß die entwickelten Teile der Sub-DAs immer korrekt zusammenpassen. Der dritte Beziehungstyp, Usage, verläuft dagegen immer zwischen einer Super-DA und ihren Sub-DAs. Dadurch wird gewährleistet, daß die Super-DA den Austausch vorläufiger Entwurfsdaten mit der Sub-DA kontrollieren kann. Es besteht jedoch nicht die Möglichkeit, daß die Sub-DA von sich aus Daten vorzeitig weitergibt. Stattdessen muß die Super-DA vorzeitige Daten ihrer Sub-DAs selber anfordern.

Innerhalb einer Design Activity wird dann für deren Aufgaben ein Workflow erstellt. Für die Start-DA ‘Konstruiere Kem’ müßte ein solcher Workflow wie in Abbildung 3.24 aussehen. Der Workflow besteht aus den Design Operations (DOPs) ‘Lege Konstruktionsumfang fest’, ‘Entwickle Teile’, ‘Prüfe Konstruktionsumfang’ und ‘Führe Freigabe durch’. Er enthält keine parallelen Zweige, durch zwei Schleifen kann jedoch bei in späteren DOPs aufgetretenen Problemen wieder weiter nach vorne im Workflow gesprungen werden. Die spezielle DOP ‘Evaluate’, die immer am Ende eines Workflow durchlaufen wird, stellt sicher, daß auch die geforderte Qualität der Entwurfsobjekte zurückgegeben wird.

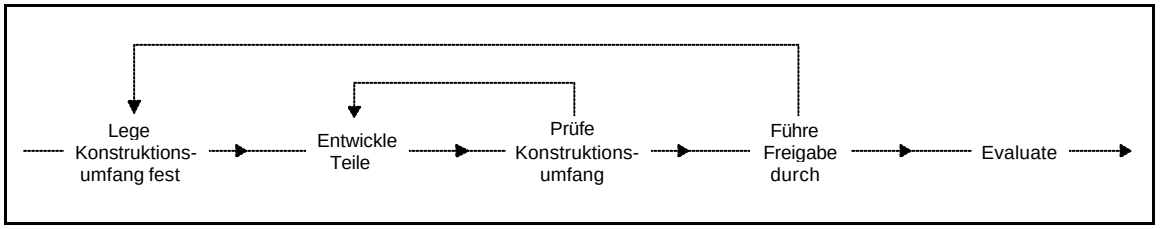


Abbildung 3.24: Workflow der DA ‘Konstruiere Kem’

Die DOP ‘Entwickle Teile’ ist die Aufgabe, von der entsprechend der DA-Hierarchie Teilaufgaben ausgegliedert werden. Für jede Design Activity, bis hinunter zur DA ‘Konstruiere Bauteil’, wird ein eigener Workflow erzeugt. Diese Workflowerzeugung geschieht mit Hilfe eines Skripts (Ausführungsplan), das mit dem DC-Parameter des Beschreibungsvektors übergeben wird.

Die Workflows in unserem Beispiel sind meist sehr klein, wie auch das nächste Beispiel zeigt (Abbildung 3.25). Es stellt den Workflow einer DA ‘Konstruiere Bauteil’ dar. Dies liegt zum einen natürlich daran, daß durch die übergeordnete Ebene der DA-Hierarchie aus der Administrations- / Kooperations-Ebene einiges, den Kontrollfluß betreffend, aus dem Workflow genommen und nun durch die Delegationsbeziehung dargestellt wird.

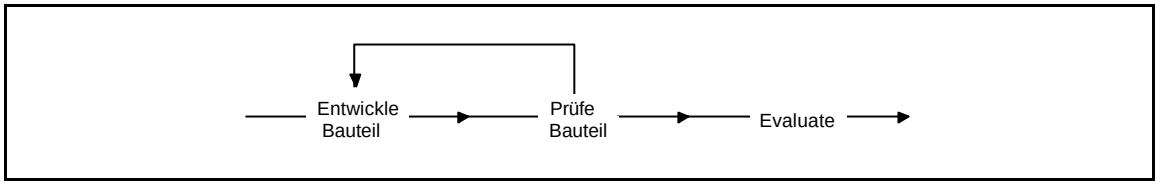


Abbildung 3.25: Workflow einer DA ‘Konstruiere Bauteil’

Innerhalb eines solchen Workflows gibt es außer dem Kontrollfluß weitere Möglichkeiten Einfluß zu nehmen. Durch Constraints läßt sich die Ausführung einer DOP beschränken. Dadurch kann beispielsweise simuliert werden, daß eine DOP ‘Prüfe Bauteil’ erst gestartet werden kann, wenn die DOP ‘Entwickle Bauteil’ eine bestimmte Qualitätsstufe eines Datenobjekts erzeugt hat.

3.2.7 Abgrenzung und Bewertung

Das Schlagwort, mit dem CONCORD versucht, seine Architektur zu erklären, ist das Designflow-Management. Man kann es eigentlich einfach als Workflow-Management bezeichnen, das jedoch noch von einer Schicht umgeben ist, die die flexible Planung von Workflows unterstützt. Das CONCORD-System umfaßt also geregelte wie völlig ungeregelte Abläufe. Es definiert dazu drei Abstraktionsebenen.

Die oberste Ebene, die Administrations- / Kooperations-Ebene erlaubt den Bearbeitern flexibel und dynamisch eine hierarchische Struktur von Aktivitäten (Design Activities) aufzubauen, indem sie Teilaufgaben in neue Aktivitäten auslagern. Auf der zweiten Ebene, der Plan-Ebene, wird der Inhalt einer Aktivität aus der ersten Ebene mittels eines strukturierten und nicht mehr änderbaren Workflows festgelegt. Designflow (mittels Design Activities) und Workflow (Inhalt der Design Activities) werden so strikt voneinander getrennt.

Das Ziel von Designflow-Management soll die Unterstützung der Bearbeiter sein, ohne dabei deren Kreativität einzuschränken. Anhand der Vorgehensweise von CONCORD, läßt sich jedoch recht trefflich darüber streiten, ob dieses Ziel immer erreicht wird. Zwar sind die Bearbeiter auf der obersten Ebene sehr frei in ihrer Kreativität, was die Anordnung der Aktivitäten angeht. Auf der zweiten Ebene, innerhalb einer Aktivität haben sie jedoch keinerlei Einfluß mehr. Wenn wir dies mit der Vorgehensweise von WEP vergleichen, so ist es dort nämlich genau umgekehrt. Setzen wir dazu erstmal eine Design Activity (DA) von CONCORD mit einer Activity von WEP gleich. Ebenso eine Design Operation (DOP) von CONCORD mit

einem Programstep von WEP. Vergleicht man nun beide Systeme, so sieht man sofort den Unterschied. Bei WEP kann der Bearbeiter innerhalb der Activity seiner Kreativität freien Lauf lassen. Er wird im Prinzip nur durch die verfügbaren Programsteps eingeschränkt. Bei WEP läuft der strukturierte Workflow außerhalb der Aktivitäten ab. Der Bearbeiter hat auf die Anordnung der Aktivitäten keinen Einfluß mehr. Für Produktentwicklungsprozesse ist dies der Normalfall. Ihre grobe Struktur, die die Zusammenarbeit verschiedener Bereiche (Abteilungen, Konstruktionsgruppen, ...) festlegt, ist vorgegeben und repräsentiert das Standardvorgehen des jeweiligen Unternehmens bei der Produktentwicklung. Ein WEP-Workflow koordiniert die involvierten Bearbeiter entsprechend dieses Standardvorgehens. Der Bearbeiter hat jedoch innerhalb der von ihm zu erledigenden Aufgabe (Aktivität) alle Freiheiten und kann selber entscheiden, wann und wie er etwas bearbeitet.

Hier kann man also behaupten, CONCORD betreibt Workflow mit einzelnen Programschritten, den Design Operations. Deren interner Ablauf ist fest vorgegeben und wird durch die dritte und unterste Ebene, die Werkzeug-Ebene, gesteuert. Der Ablauf entspricht dem einer Transaktion. WEP betreibt dagegen Workflow mit Aktivitäten. Die Programschritte sind innerhalb der Aktivität und werden erst vom Bearbeiter aufgerufen.

Neben diesen Unterschieden hat CONCORD aber auch einige Ansätze, die denen von WEP recht ähnlich sind. Dies betrifft zum Beispiel den Beziehungstyp Usage. Er wird in der DA-Hierarchie dort eingetragen, wo eine DA vorzeitige Daten von einer anderen DA anfordern kann. Dazu gibt es zwei spezielle Funktionen. *Require* fordert von einer anderen DA Daten an. Dadurch wird die Beziehung aufgebaut. *Propagate*, von der aufgeforderten DA aufgerufen, gibt die Daten für die anfordernde DA frei. So ist die Möglichkeit gegeben, daß mehrere DAs beziehungsweise Bearbeiter an demselben Datenobjekt arbeiten (Simultaneous Engineering). Dies läßt sich mit der benutzerinitiierten Weitergabe vorzeitiger Daten bei WEP vergleichen. Hier gibt es die entsprechenden Funktionen *RequestObject* und *ReleaseObject.* Bei CONCORD existiert diese Beziehung aber grundsätzlich nur im Designflow. Im Workflow besteht keine entsprechende Unterstützung der vorzeitigen Datenweitergabe. Auch muß beachtet werden, daß nur über diesen Mechanismus das gleichzeitige Arbeiten auf demselben Datenobjekt erreicht werden kann. Es besteht nicht die Möglichkeit, daß eine DA ihre Daten von sich aus vorzeitig freigibt.

Eine weitere Ähnlichkeit zwischen CONCORD und WEP findet sich im Bereich Gruppenarbeitsmechanismen. So erlaubt der Beziehungstyp Negotiation das Verhandeln von Zielspezifikationen zwischen Sub-DAs. Super-DAs sind dabei jedoch ausgeschlossen. Der Gegenstand dieser Kooperation ist eine Entwurfszielspezifikation, also der SPEC-Parameter des Beschreibungsvektors einer DA. So kann eine Sub-DA einer anderen verfeinerte oder neue Eigenschaften für deren Spezifikation vorschlagen. Dazu dient die Funktion *Propose*. Die angesprochene DA kann die Vorschläge annehmen (*Agree*) oder ablehnen (*Disagree*). WEP bietet für diesen Zweck die Abstimmungsrunde (Consolidation) an. Hier gibt es auch wieder entsprechende Funktionen: *ProposeNewObject, AgreeNewObject* und *RejectNewObject.* Bei WEP gibt es jedoch keine Einschränkung, welche Aktivitäten an einer solchen Abstimmungsrunde teilnehmen dürfen, sofern sie alle auf demselben Objekt arbeiten. Der Hauptunterschied zwischen beiden Systemen ist aber wohl das, über was man sich in einer Abstimmungsrunde einigen muß. Bei CONCORD wird über eine Spezifikation, das heißt über das Ziel der zu bearbeitenden Aufgabe abgestimmt. Bei WEP wird direkt über eine Objektversion abgestimmt, wobei die Möglichkeit besteht, über Objekte mit beliebigem Inhalt abzustimmen. Weiterhin hat bei WEP der ‘Besitzer’ des Objekts immer noch die Möglichkeit, das Ergebnis der Abstimmungsrunde nicht zu beachten. Bei CONCORD muß im Falle, daß man sich nicht einigen kann, der Bearbeiter der Super-DA eine Entscheidung treffen.

CONCORD bietet zur Unterstützung von Produktentwicklungsprozessen einige interessante Ansätze. Speziell was die Bereiche Simultaneous Engineering und Gruppenarbeits-mechanismen

angeht. Die flexible Erstellung eines Designflow und der innerhalb einer Design Activity auf Programmschritten ablaufende Workflow ist aber in der Praxis für die Produktentwicklung eher nicht geeignet. Was CONCORD außerdem bisher fehlt, ist ein Zeitmanagement. Dadurch können Design Activities natürlich keine Zeitangaben zugewiesen werden. Für die Produktentwicklung ist dies jedoch unerläßlich, schließlich müssen Fertigungstermine eingehalten werden.

3.3 DYNAMITE

DYNAMITE (**DYNAMI**c **T**ask n**E**ts) ist Teil eines umfassenden Forschungsgebiets, das sich mit der Entwicklung einer Umgebung beschäftigt, die auf administrativer Ebene versucht, die Verwaltung von Produkten, Prozessen und Hilfsmitteln zu integrieren (vgl. [NaWe94]).

DYNAMITE formalisiert einen Teil dieses Modells (vgl. [HJKW96], [HJKW97]). Dabei geht es speziell um die Modellierung und Ausführung von Aufgaben, respektive Prozessen, und den zu ihrer Realisierung durchzuführenden Schritten. Dies erfolgt unter einer administrativen Perspektive mit Hilfe von sich dynamisch entwickelnden Aufgabennetzen. Zur Beschreibung dient ein Prozeß-Meta-Modell, dem ein netz- beziehungsweise graphenbasierter Formalismus (PROGRES) zugrundeliegt.

Die Hauptzielsetzung von DYNAMITE ist die Unterstützung des Prozesses der Software-erstellung und -wartung. Komplexe Softwareprozesse können nur selten im voraus geplant werden. Um solche Prozesse ohne Einschränkung der Prozeßdynamik verwalten und leiten zu können, muß als Grundvorraussetzung die Unterstützung für folgende Punkte gegeben sein:

- **Fortschreitende Entwicklung**: Die Prozeßstruktur ist von der Produktstruktur abhängig, die sich nach und nach weiterentwickelt.
- **Simultane Bearbeitung**: Darunter versteht man die Erweiterung der Kooperation zwischen Softwareentwicklern, um sobald als möglich vernünftige zwischenzeitliche Ergebnisse liefern zu können.
- **Rückgriff**: In späteren Phasen entdeckte Fehler erfordern eine Ergebnisverbesserung früherer Phasen.

Der DYNAMITE-Ansatz auf Basis der dynamischen Aufgabennetze wurde entwickelt, um diese Anforderungen zu erfüllen.

Die dynamischen Aufgabennetze von DYNAMITE werden formal in PROGRES definiert. PROGRES (**PRO**grammierte **GR**aphen-**E**rsetzungs-**S**ysteme) ist ein Werkzeug zur ‘programmierten Graphenersetzung’. Eine Spezifikation in PROGRES besteht immer aus zwei Teilen: einem Graphen und einer oder mehreren Graphenersetzungsregeln. Der **Graph** definiert Knoten, Kanten und Attribute. Eine **Graphenersetzungsregel** beschreibt die Ersetzung eines gesamten Sub-Graphen in einer verständlichen Weise. Besser als es mit elementaren Operationen, wie dem Erstellen oder Löschen von einzelnen Knoten und Kanten möglich wäre. Aus einer PROGRES-Spezifikation läßt sich in der PROGRES-Entwicklungs-umgebung ein interaktives System erzeugen, mit dem der Graph editiert und mit Hilfe der Ersetzungsregeln transformiert werden kann. Eine weitergehende Betrachtung von PROGRES wäre für diese Beschreibung des DYNAMITE-Ansatzes zu umfangreich. Mehr Informationen zu PROGRES können jedoch in [SWZ95] nachgelesen werden.

3.3.1 Struktur eines Aufgabennetzes

Das zentrale Modellierungselement in DYNAMITE ist die **Aufgabe** (Task). Sie beschreibt eine zu erledigende Arbeit. Die Beschreibung einer Aufgabe wird unterteilt in eine Schnittstelle und gegebenenfalls mehrere Realisierungen.

Die **Schnittstelle** (Interface) beschreibt das ‘was’ der Aufgabe. Sie legt die Aufgabenstellung, die Ein- und Ausgabeparameter, Vor- und Nachbedingungen und Startzeiten bezüglich des Ausführungsverhaltens fest und beschreibt damit, was zu erstellen ist. Dabei dient sie als Abstraktion für die Realisierung, da eine Aufgabe generell auf verschiedene Arten (Realisierungen) ausgeführt werden kann.

Eine **Realisierung** gibt dagegen die Aktionen an, die für die Erfüllung der Aufgabe durchzuführen sind. Sie legt somit fest, wie vorzugehen ist. Eine Realisierung kann entweder atomar oder komplex sein. Im Falle einer **atomaren** Realisierung wird die Aufgabe nicht in Unteraufgaben zerlegt, sondern die Vorgehensweise wird einfach durch eine informelle Beschreibung festgelegt. Ein Beispiel wäre das Editieren oder Compilieren eines Software-Moduls. Eine **komplexe** Realisierung besteht aus einem Aufgabennetz, welches Unteraufgaben enthält, die durch verschiedene Beziehungen verbunden sind. Eine komplexe Realisierung spannt somit eine Aufgabenhierarchie auf.

Im folgenden werden die möglichen Beziehungen beschrieben, die die Unteraufgaben in einem Aufgabennetz verbinden. Dabei wird zwischen Kontrollfluß- und Datenfluß-beziehungen unterschieden. Kontrollflüsse steuern die Ausführung, während die Datenflüsse für den Datenaustausch zwischen den Unteraufgaben sorgen.

Kontrollflußbeziehungen definieren eine Reihenfolgeabhängigkeit auf den Unteraufgaben. Sie spannen einen azyklischen Graphen auf, der als Skelett für das Aufgabennetz dient. Neben der normalen Kontrollflußbeziehung, die immer in die Ablaufrichtung des Prozesses verweist, enthält das Meta-Modell noch eine **Rückgriffbeziehung**, mit der sich Rückgriffe und Zyklen im Aufgabennetz beschreiben lassen. Sie ist entsprechend in die entgegengesetzte Richtung orientiert. Muß eine abgeschlossene Aufgabe anhand einer Rückgriffbeziehung oder neuer Eingabedaten wieder reaktiviert werden, so wird von ihr eine neue Aufgabenversion erstellt. Aufgabenversionen sind über **Versionsbeziehungen** verbunden.

Abbildung 3.26 zeigt ein Beispiel eines Aufgabennetzes und die verschiedenen Kontrollfluß-beziehungen. Das Aufgabennetz beschreibt den Prozeß einer Softwareentwicklung. Die Software besteht aus den vier Modulen A bis D. Sie beginnt immer mit dem Design und endet mit einem Systemtest. Die Design-Aufgabe wird gefolgt von der gleichzeitigen Implementierung aller Module. Für jedes Modul wird außerdem ein Modultest durchgeführt. Der abschließende Systemtest stellt die korrekte Integration der Module sicher. Zu erkennen ist, daß die Module B und C vom Modul A Daten importieren, also von ihm abhängig sind. Dasselbe gilt für Modul D, bezüglich der Module B und C.

Die Kontrollflußhierarchie ergibt sich durch die gerichteten Kontrollflußkanten. Die Rückgriffbeziehung verläuft, wie bereits beschrieben, genau entgegengesetzt. Am Beispiel verläuft sie vom abschließenden Systemtest zurück zur Aufgabe *Implement_B*. Neue Aufgabenversionen werden grau unterlegt dargestellt. Sie wurden im Beispiel für die drei Aufgaben *Implement_B*, *Test_B* und *Test_D* erzeugt. Eine genauere Beschreibung dieser Aktion findet sich in Kapitel 3.3.4.

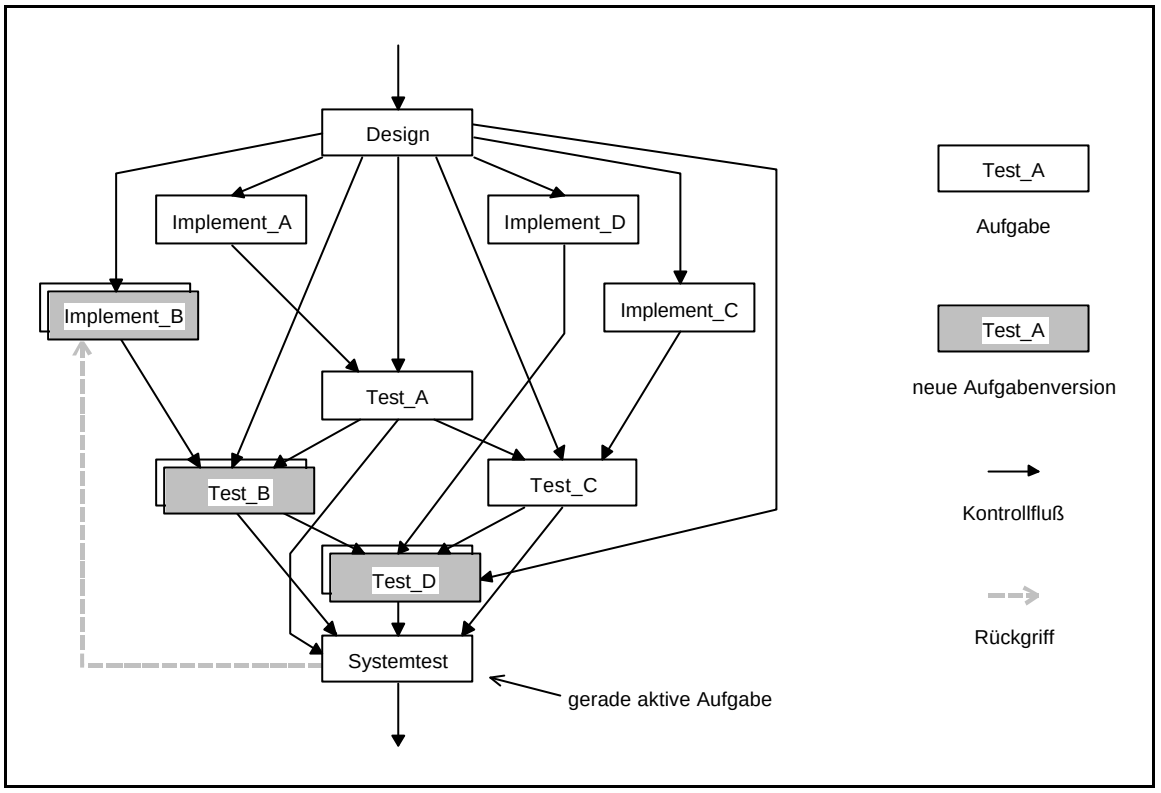


Abbildung 3.26: Beispiel für ein Aufgabennetz mit Darstellung des Kontrollflusses

Während mit dem Kontrollfluß die Beziehungen zwischen den Aufgaben modelliert werden, verbinden die **Datenflußbeziehungen** die Ein- und Ausgabeparameter der Aufgaben. Es gibt drei Arten von Datenflußbeziehungen:

- **Eingabe → Eingabe**: Die Eingabe einer übergeordneten Aufgabe wird zu einer Unteraufgabe weitergeleitet.
- **Ausgabe → Eingabe**: Normaler Fall, die Ausgabe einer Unteraufgabe wird zur Eingabe einer anderen Unteraufgabe.
- **Ausgabe → Ausgabe**: Die Ausgabe einer Unteraufgabe wird an die übergeordnete Aufgabe zurückgegeben.

Der Datenfluß verläuft also nicht nur horizontal zwischen den Unteraufgaben eines Aufgabennetzes, sondern auch vertikal zwischen Aufgabe und Unteraufgabe.

Abbildung 3.27 illustriert die Datenflußbeziehungen an einem Ausschnitt des Aufgabennetzes von Abbildung 3.26. Die Aufgabe *Design* erhält von der ihr übergeordneten Aufgabe die Startdefinitionen und Zielvorgaben (*Sys.req*). Daraufhin erstellt sie die Schnittstellen-definitionen für alle Module (*X.int*). Die Implementierungsaufgabe für Modul B (*Implement_B*) bekommt als Eingabedaten die Schnittstelle des zu implementierenden Moduls (*B.int*). Ein zu implementierendes Modul wird als Export-Modul bezeichnet. Da Modul B, wie bereits erwähnt, von Modul A (Import-Modul) abhängt, ist auch dessen Schnittstelle (*A.int*) als Eingabe erforderlich. Daraufhin kann *Implement_B* eine Implementierung von Modul B (*B.impl*) erstellen. Die Test-Aufgabe für Modul B (*Test_B*) erhält die Implementierung von B und eine getestete Implementierung von Modul A (*A.tst*), worauf sie in der Lage ist, die eigene Implementierung zu testen.

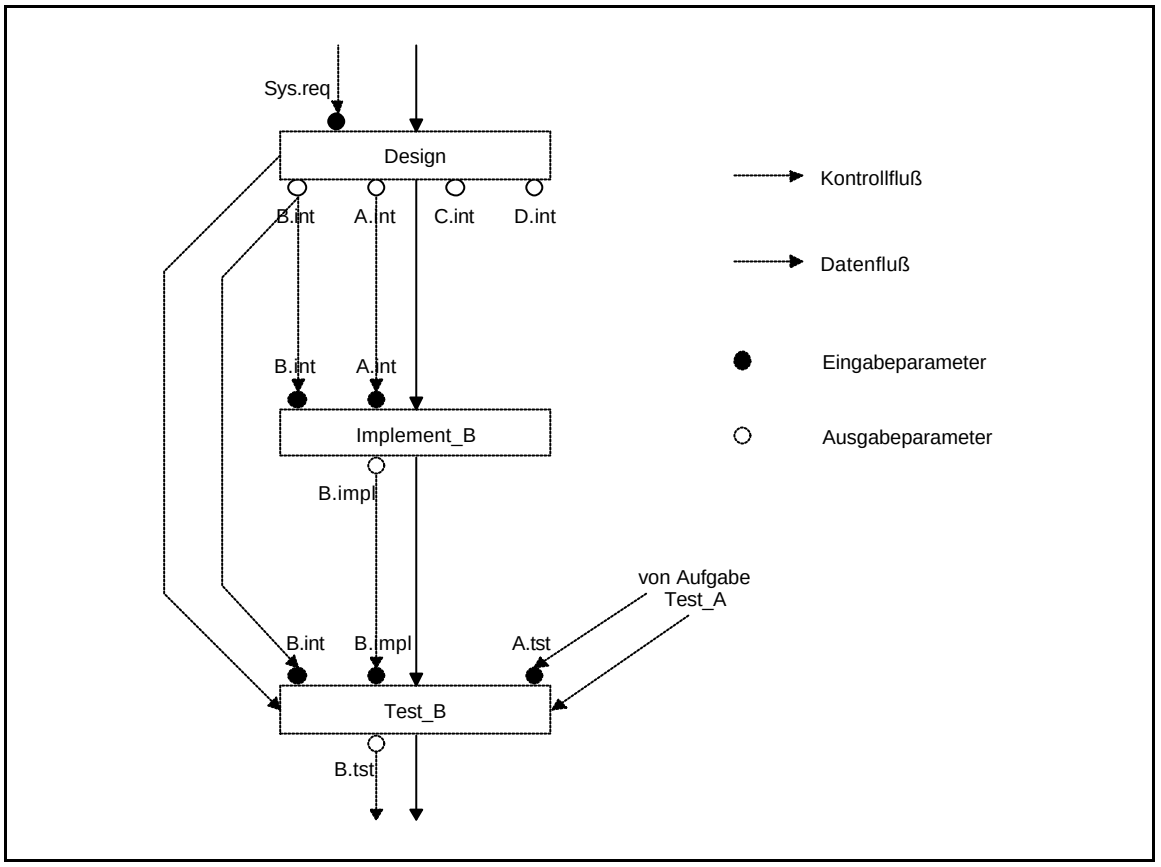


Abbildung 3.27: Beispiel für den Datenfluß in einem Ausschnitt eines Aufgabennetzes

3.3.2 Modellierungsebenen

Für die Repräsentation von dynamischen Aufgabennetzen im Computer wird zwischen drei Modellierungsebenen unterschieden, dem generischen Modell, dem spezifischen Modell und der Instanz-Ebene.

Das **generische Modell** (Schema) ist unabhängig vom Anwendungsbereich. Auch wenn sich DYNAMITE hauptsächlich auf die Softwareentwicklung konzentriert, kann ein Schema prinzipiell auch auf andere Bereiche wie die Computerunterstützte Fertigung (Computer Integrated Manufacturing - CIM) oder Büroautomation angewandt werden. Es bestimmt die allgemeinen Merkmale eines dynamischen Aufgabennetzes für verschiedene Anwendungsbereiche. Dabei legt es in abstrakter Form Begriffe wie Aufgabe, Eingabe, Ausgabe, Datenfluß usw. fest, und stellt ein Standard-Zustands-Übergangsdiagramm zur Verfügung, das in der nächsten Ebene angepaßt werden kann.

Das **spezifische Modell** paßt ein generisches Modell an einen bestimmten Anwendungsbereich an. Abbildung 3.28 zeigt den strukturellen Teil (Prozeßstruktur-Schema) eines spezifischen Modells für das bereits als Beispiel dargestellte Aufgabennetz (Abbildung 3.26). Aufgabentypen werden durch Ellipsen repräsentiert. Für jeden Aufgabentyp ist die minimale und maximale Anzahl Instanzen angegeben, die während der Ausführung des Aufgabennetzes erzeugt werden können. Die Spezifikationen der Eingabe- und Ausgabeparameter bestehen jeweils aus dem Namen, dem Datentyp und der minimalen und maximalen Anzahl Parameter dieses Typs, die

eine Aufgaben-Instanz besitzen kann. Für die Rückgriffsbeziehungen wurden ebenfalls passende Parameter definiert, die aber nicht unbedingt angegeben werden müssen. Es handelt sich dabei um eine in Textform angegebene Beschreibung des Grundes für den Rückgriff.

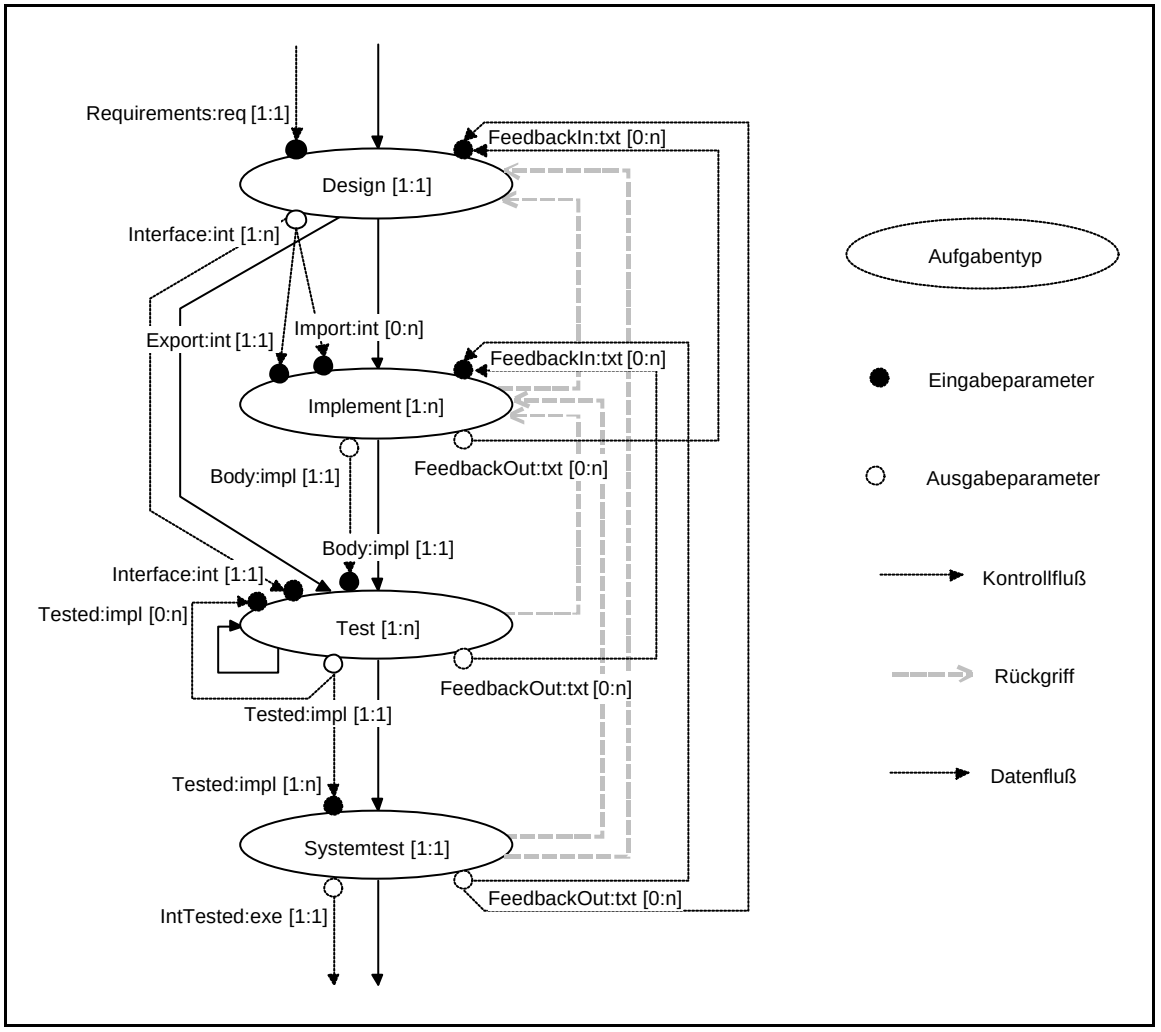


Abbildung 3.28: Prozeßstruktur-Schema des Spezifischen Modells für das Beispiel-Aufgabennetz

Auf der **Instanz-Ebene** werden während der Ausführung die konkret durchzuführenden Aufgaben als Ausprägung des Aufgabennetzes erstellt. Im Aufgabennetz werden die Aufgaben dabei immer mit ihrem aktuellen Status dargestellt (siehe Kapitel 3.3.3).

3.3.3 Modellierung des Ausführungsverhaltens

Die Ausführungssemantik von DYNAMITE wurde speziell für die Unterstützung der dynamischen Aufgabennetze definiert. Aufgabennetze müssen zu Beginn nicht komplett modelliert sein, sondern können während der Ausführung inkrementell erweitert werden. Unteraufgaben, die von einer Änderung nicht betroffen sind, können so während der Modifikation ohne Unterbrechung fortgeführt werden. Dies schließt allerdings eine Unterbrechung des gesamten Aufgabennetzes bei großen Änderungen nicht aus.

Ein vordefiniertes **Zustands-Übergangsdiagramm** legt die Prozeßzustände der Aufgaben und ihre Übergänge fest und bildet die Grundlage für die Festlegung der Ausführungssemantik (Abbildung 3.29).

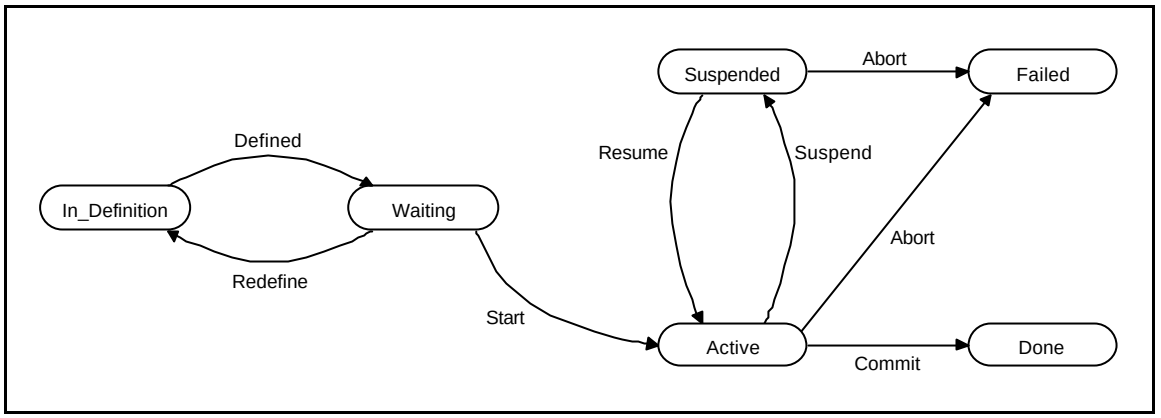


Abbildung 3.29: Zustands-Übergangsdiagramm von DYNAMITE

Der initiale Zustand einer gerade neu erstellten Aufgabe ist *In_Definition*. Hier wird die Schnittstelle der Aufgabe beschrieben (Zuweisung der Ein- und Ausgaben, etc.). Ist die Aufgabendefinition beendet, ändert sie ihren Zustand über den Übergang *Defined* in *Waiting*. Die Aufgabe ist nun bereit, ausgeführt zu werden. Der inverse Übergang *Redefine* kann benutzt werden, um Änderungen an der Aufgabendefinition vorzunehmen. Der Übergang *Start* leitet die Aufgabe weiter in den Zustand *Active*. Die Ausführung der Aufgabe kann über den Übergang *Suspend* zeitweise ausgesetzt (Zustand *Suspended*) und mittels *Resume* wieder fortgeführt werden. Das Diagramm besitzt zwei finale Zustände. *Done* zeigt an, daß die Aufgabe erfolgreich beendet wurde (Übergang *Commit*). *Failed*, über einen *Abort*-Übergang, legt die Ausführung als nicht erfolgreich beendet fest.

Das Übergangsdiagramm ist ein fester Bestandteil des DYNAMITE-Modells und kann bei der Anpassung eines spezifischen Modells an einen Anwendungsbereich nicht geändert werden. Geändert werden, können aber die Bedingungen, die zu einem Zustandübergang führen. So ist jeder Übergang zu Beginn mit einer Standard-Bedingung versehen, die aber für jeden Aufgabentyp individuell definiert werden kann. Beispielsweise kann der *Start*-Übergang von den Vorgänger-Aufgaben im Kontrollfluß anhängig gemacht werden. So kann der Übergang einerseits nur gestattet sein, wenn alle vorhergehenden Aufgaben erfolgreich beendet wurden. Andererseits kann er auch bedingen, daß die vorhergehenden Aufgaben vorzeitige Daten liefern.

3.3.4 Simultanes Arbeiten und Rückgriffe

DYNAMITE unterstützt simultanes Arbeiten (Concurrent Engineering), indem es die Ausführung mehrerer Aufgaben gleichzeitig ermöglicht. So ist es im Beispiel in Abbildung 3.26 möglich, daß die Implementierungsaufgaben vor Beendigung der Design-Aufgabe gestartet werden. Abbildung 3.30 zeigt einen Teil des dort beschriebenen Aufgabennetzes.

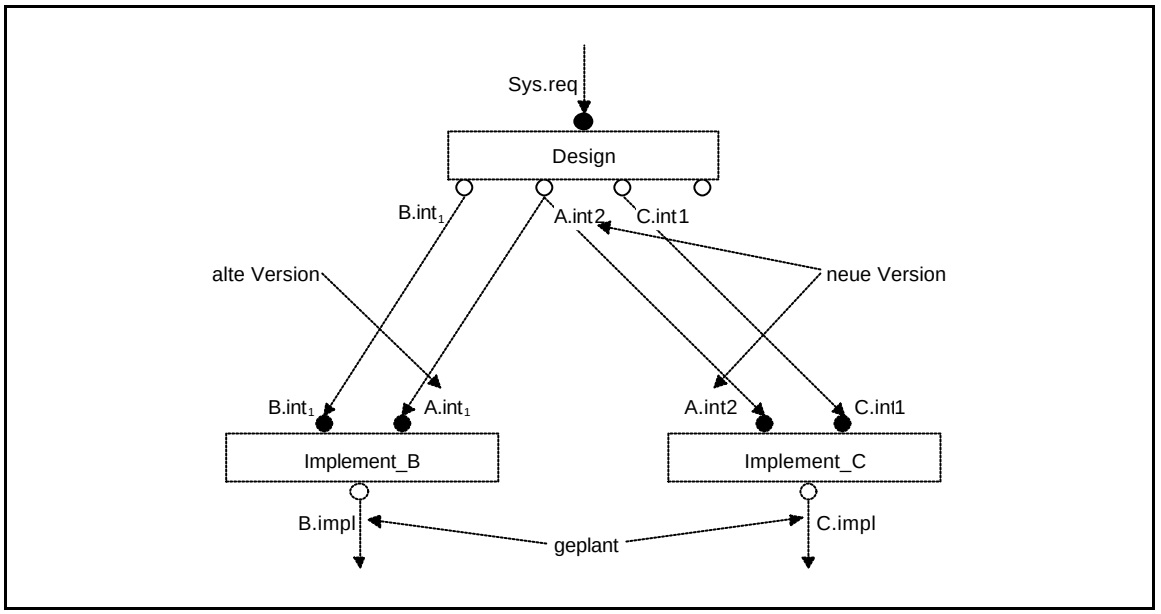


Abbildung 3.30: Verschiedene Objektversionen an einem Ausschnitt des Beispiel-Aufgabennetzes

Während der Ausführung kann eine aktive Aufgabe verschiedene Versionen ihrer Ausgabeobjekte erzeugen und ebenfalls verschiedene Versionen der Eingabeobjekte verarbeiten. Das System muß darüber Buch führen, welche Aufgaben welche Versionen erstellt und benutzt haben. In Abbildung 3.30 liefert die Design-Aufgabe eine vorzeitige Version der Schnittstellendefinition von Modul A ($A.int_1$). Diese Version kann von den Implementierungsaufgaben für die Erstellung der Module B und C verwendet werden, worauf diese starten. Nun sind alle drei dargestellten Aufgaben aktiv. Zu einem späteren Zeitpunkt liefert die Design-Aufgabe die fertige Version der Schnittstellendefinition von Modul A ($A.int_2$). Die jetzige Situation ist in der Abbildung eingefroren. *Implement_C* hat die neue Version gelesen, *Implement_B* arbeitet jedoch weiter auf der alten Version. Um die daraus entstehenden Inkonsistenzen zu vermeiden, müssen die Bedingungen des Zustandsübergangs Commit sicherstellen, daß *Implement_B* in dieser Situation nicht erfolgreich abgeschlossen werden kann.

Komplexe Entwicklungen laufen nicht immer glatt ab. Wenn eine Aufgabe einen Fehler in einem Eingabeobjekt entdeckt, wird eine Rückgriffbeziehung in das Aufgabennetz eingebunden. Über diese Beziehung wird der Fehler dem Ersteller des inkorrekten Objekts gemeldet. Über die Aufgabenstruktur des Spezifischen Modells (Abbildung 3.28) wird kontrolliert, daß nur erlaubte Rückgriffbeziehungen im Aufgabennetz eingetragen werden. Verläuft ein Rückgriff zu einer aktiven Aufgabe (im Falle des Concurrent Engineering), so kann sie diesen direkt behandeln, indem sie eine neue korrigierte Objektversion erstellt. Ist die Aufgabe dagegen bereits abgeschlossen, so wird es etwas komplizierter. Schließlich kann es sein, daß bisher verwendete Ressourcen oder Bearbeiter der Aufgabe nicht mehr verfügbar sind. In diesem Fall wird, anstatt die Aufgabe zu reaktivieren, eine neue Aufgabenversion erzeugt. Dies hat mehrere Vorteile. So kann die neue Aufgabenversion gegebenfalls mit einem anderen Bearbeiter versehen werden oder eine geänderte Aufgabenstellung erhalten. Ein weiterer Vorteil ist die Nachvollziehbarkeit: Die zuvor durchgeführten Aufgaben bleiben nach dem Erzeugen einer neuen Aufgabenversion erhalten. Dadurch läßt sich der Vorgang, der zum Fehler geführt hat, später nachvollziehen. Die Eingaben für die neue Aufgabenversion sind die Informationen der Rückgriffbeziehung und Kopien der Eingaben der ursprünglichen Aufgabenversion. Nachfolgende Aufgaben, die vom korrigierten Ausgabeobjekt abhängen, werden automatisch informiert. Sind sie bereits abgeschlossen, so müssen sie ebenfalls als neue Aufgabenversion gestartet werden.

3.3.5 Abgrenzung und Bewertung

In den Beschreibungen zu DYNAMITE tauchen nirgendwo die Begriffe Workflow oder Workflow-Management auf. Dennoch kann man es sicherlich am einfachsten unter dem Bereich Workflow-Modelle einordnen. Betrachtet man die Aufgaben von DYNAMITE als Aktivitäten eines Workflow-Management-Systems und ein Aufgabennetz als Modellierung eines Prozesses, so ergeben sich doch viele Ähnlichkeiten.

Der normale Kontrollfluß legt die Reihenfolge der Prozeßschritte (Aktivitäten, Aufgaben) fest. Eine Rückgriffbeziehung kann als Schleifenkonstrukt bezeichnet werden. Sie wird jedoch immer nur einmal durchlaufen und danach wieder abgebaut. Sollte später an gleicher Stelle wieder ein Fehler auftreten, wird sie neu definiert. Der Datenfluß beschreibt die Abhängigkeiten zwischen den Eingaben und Ausgaben der Prozeßschritte. Beziehungen zu globalen Objekten und eine Objektverwaltung werden in DYNAMITE jedoch nicht beschrieben.

Der größte Unterschied zu Workflow-Management-Systemen ist der grundsätzlich dynamische Aufbau eines Aufgabennetzes. Workflow-Management-Systeme basieren auf einem vormodellierten Kontrollfluß. In DYNAMITE existiert am Anfang nur ein Prozeßstruktur-Schema (vgl. Abbildung 3.28), das als Vorgabe für den Aufbau eines Aufgabennetzes dient. Es legt die möglichen Aufgaben, Ein- und Ausgaben der Aufgaben, Kontrollflüsse und Datenflüsse fest. Aufbauend auf diesen Vorgaben kann sich ein Aufgabennetz einer komplexen Aufgabe dynamisch entwickeln. Dabei ist der Aufbau und die Ausführung eines Aufgabennetzes verschränkt. Das bedeutet, das Netz kann sich weiterentwickeln, während einige Aufgaben bereits ausgeführt werden. Dieses Vorgehen ermöglicht eine schrittweise Evolution eines Aufgabennetzes. Allerdings muß dabei die Entwicklung eines Sub-Netzes einer komplexen Aufgabe in jedem Durchlauf aufs neue erfolgen, dies bedingt einen noch nicht näher spezifizierbaren Aufwand.

In Bezug auf Produktentwicklungsprozesse bietet DYNAMITE einige interessante Ansätze. Die dynamische Entwicklung des Aufgabennetzes liefert die Unterstützung für eine variable Anzahl von Objekten pro Durchlauf. Dazu trägt auch die Aufgabenverwaltung bei. Sie ist generell so ausgelegt, daß mehrere Aufgaben gleichzeitig aktiv sein können. So sind parallele Kontrollflüsse, ebenso wie eine vorzeitige Datenweitergabe realisierbar. Sollten in späteren Aufgaben durch fehlerhafte vorzeitige Daten Fehler auftreten, so kann dies über eine Rückgriffbeziehung zurückgemeldet werden. Ist die Aufgabe, die den Fehler verursacht hat noch aktiv, kann sie sofort eine korrigierte Version liefern. Wie genau ein solcher Vorgang abläuft, ist jedoch in DYNAMITE bislang nicht spezifiziert. Dies betrifft beispielsweise auch die Einarbeitung neuer Versionen. So ist nicht geklärt, ob eine Aufgabe beim Eintreffen einer neuen Eingabeversion abgebrochen und vollständig neu gestartet werden muß, oder ob eine Möglichkeit der Einarbeitung besteht.

Ein Problem kann auch das Fehlen einer direkten Unterstützung unstrukturierter Teilprozesse sein. In DYNAMITE wird zwischen atomaren Aufgaben und komplexen Aufgaben unterschieden. Eine atomare Aufgabe kann mit der Ausführung eines einzelnen Werkzeugs verglichen werden. Eine komplexe Aufgabe wird in Unteraufgaben zerlegt. Der Ablauf solcher Unteraufgaben muß aber wiederum über ein Aufgabennetz dargestellt werden. Eine völlig freie Arbeitsreihenfolge kann also nicht modelliert werden. Eventuell liese sich eine derartige Flexibilität durch ein Prozeßstruktur-Schema erreichen, das durch den Kontrollfluß alle Möglichkeiten der Arbeitsreihenfolge offen läßt. Mit der Anzahl zur Verfügung stehender Werkzeuge steigt dabei jedoch auch der Umfang und die Unübersichtlichkeit eines solchermaßen gerüsteten Prozeßstruktur-Schemas sehr schnell.

Problematisch kann auch das Fehlen einer Beziehung sein, die auf Aufgaben außerhalb der Grenzen eines Aufgabennetzes verweisen kann. Die Sub-Netze von komplexen Aufgaben in DYNAMITE sind vollständig abgeschlossen. Sie bekommen ihre Eingaben von der übergeordneten Aufgabe und liefern ihre Ausgaben auch dort wieder ab. Ein Informationsaustausch, beispielswiese nur zur Verständigung zweier Bearbeiter in unterschiedlichen Sub-Netzen, ist so ausgeschlossen. Er kann nur über externe Systeme realisiert werden.

Zur perfekten Unterstützung von Produktentwicklungsprozessen fehlen einem DYNAMITE-System darüberhinaus einige spezielle Elemente. So werden keine Angaben über eine Objektverwaltung für komplexe und strukturierte Objekte gemacht. Auch gibt es bislang keine Unterstützung für ein Zeitmanagement, das bei Entwicklungsprozessen zur Einhaltung von Fristen aber erforderlich ist.

DYNAMITE wurde eigentlich für den Bereich der Softwareentwicklung entworfen. Durch sein sehr allgemeines dynamisches Modell bietet es sich aber auch für viele andere Bereiche an. Der Einsatz und die Anwendbarkeit in der Praxis müssen aber erst noch überprüft werden.

3.4 Procura

Procura ist ein **Projekt-Management-Modell**, das versucht, die sonst in diesem Bereich übliche starre Planung von Prozeßabläufen flexibler zu gestalten. Die Neuerungen von Procura (vgl. [Gold96]), in Kontrast zu traditionellen Projekt-Management-Systemen, umfassen die Integration von Design, Planung und Ausführung, die Unterstützung laufender Änderungen an der Aufgaben- und Zeitplanung und ein Meldungssystem, das von einer Änderung betroffene Agenten (Benutzer und Computer, siehe unten) über die Änderung in Kenntnis setzt.

Procura zerlegt bei der Planung einen Prozeß mittels hierarchischer Baumstruktur in Teilprozesse und weist ihnen dann die benötigten Hilfsmittel, sowie den Start- und Endzeitpunkt zu. Entstehen in einem gerade ausgeführten Teilprozeß nicht erwartete Ergebnisse, so wird der noch nicht ausgeführte Teil des Prozeßablaufs davon benachrichtigt und entsprechend angepaßt. Damit ersetzt Procura die starre Hintereinanderausführung von **Aufgabenplanung** (Planning; Zerlegung in Teilprozesse), **Zeitplanung** (Scheduling; Zuordnung von Hilfsmitteln und Start- und Endzeiten) und **Prozeßausführung** (Plan Execution) in eine flexible Nebeneinnanderausführung. Die Arbeitsschritte können also, bei unterschiedlichen Teilprozessen, simultan ablaufen und sich auch rückwirkend beeinflussen.

Ein weiterer Unterschied zu herkömmlichen Projekt-Management-Systemen liegt in der Architektur von Procura. Projekt-Management-Systeme basieren im Normalfall auf einem Client-Server-System. Procura's Architektur dagegen ist Agenten-basiert. Als Agent werden dabei alle am gesamten Prozeß beteiligten menschlichen Bearbeiter und Computerprogramme bezeichnet. Zur Kommunikation zwischen diesen Agenten ist ein Satz von Nachrichten definiert. So kann recht einfach ein weiterer Agent dem System hinzugefügt werden. Das einzige, was er benötigt, ist ein Interface, welches das Nachrichtenprotokoll versteht und in die eigenen Routinen ummappt.

Zusammenfassend kann man sagen, Procura versucht die folgenden Probleme zu lösen:

- Die meisten Bereiche einer „realen Welt“ lassen sich nicht einfach und strikt formal vorplanen.
- Änderungen an der Unterteilung in Teilprozesse und an der Zeitplanung sind oftmals notwendig und sollten entsprechend unterstützt werden.
- Die Aufgabenplanung, Zeitplanung und Prozeßausführung sind voneinander abhängig und sollten deshalb überlappend abgearbeitet werden.

Die Verfahren von Procura zur Lösung dieser Probleme arbeiten dabei mit zwei elementaren Prinzipien. Die Zeitplanung (Scheduling), also das Zuordnen von Hilfsmitteln und Start- und Endzeitpunkten zum Prozeß, ist immer nur eine mehr oder weniger sichere Vorhersage und muß überprüft und gegebenenfalls abgeändert werden. Dies geschieht durch einen Vergleich der ursprünglich zugeordneten Hilfsmittel und Zeitangaben mit den aktuellen Daten der Prozesse. Der zweite Punkt ist die daraus resultierende Notwendigkeit eines Planungsmodells, das die erforderlichen Vergleiche erlaubt und Benachrichtigungen über Änderungen an betroffene Agenten weitergibt.

Procura versucht dabei nicht den Planungsprozeß zu automatisieren, sondern stellt sozusagen nur einen Buchführungsmechanismus zur Verfügung, der die überlappende Planung und Ausführung unterstützt und die notwendigen Änderungen beziehungsweise eine Neuplanung vereinfacht. Die Entscheidung, wer welchen Prozeß bearbeiten soll und wann, muß vom menschlichen Bearbeiter getroffen werden. Eine Neuzuordnung der Hilfsmittel und der Start- und Endzeitpunkte kann jedoch ebenso wie das Weiterreichen von Änderungen teilweise automatisch geschehen.

3.4.1 Modell

Procura erlaubt dem Benutzer die Ziele anzugeben, die für ein bestimmtes Projekt erreicht werden müssen. Ein solches Ziel wird allgemein **Goal** genannt. Eine **Task** (Aufgabe) beschreibt die Arbeit, die zu erledigen ist, um das Ziel zu erreichen. Dazu zählt auch die Zeitdauer, die für die Arbeitsausführung benötigt wird und die eventuelle Aufsplittung in Teilprozesse. Tasks erzeugen Ausgaben, die wiederum als Eingaben für andere Tasks dienen. Diese Eingaben und Ausgaben werden erstmals 'vorhergesagt' und müssen später überprüft werden, indem sie mit den aktuellen Eingaben und Ausgaben verglichen werden, die bei der Abarbeitung der Tasks entstehen. Eingaben und Ausgaben treten in der Form von Parametern und Eigenschaften auf und können einfache Daten wie auch physikalische Objekte repräsentieren. Tasks dienen zur Erstellung des Aufgabenplans. Ein **Scheduling Goal** korrespondiert mit einer Task und wird durch die Zuordnung von Ressourcen zu dieser Task bestimmt. Diese Ressourcen beinhalten die zugewiesenen Hilfsmittel und Bearbeiter sowie die Start- und Endzeitpunkte für den Prozeß. Mittels der Scheduling Goals wird der Zeitplan erstellt.

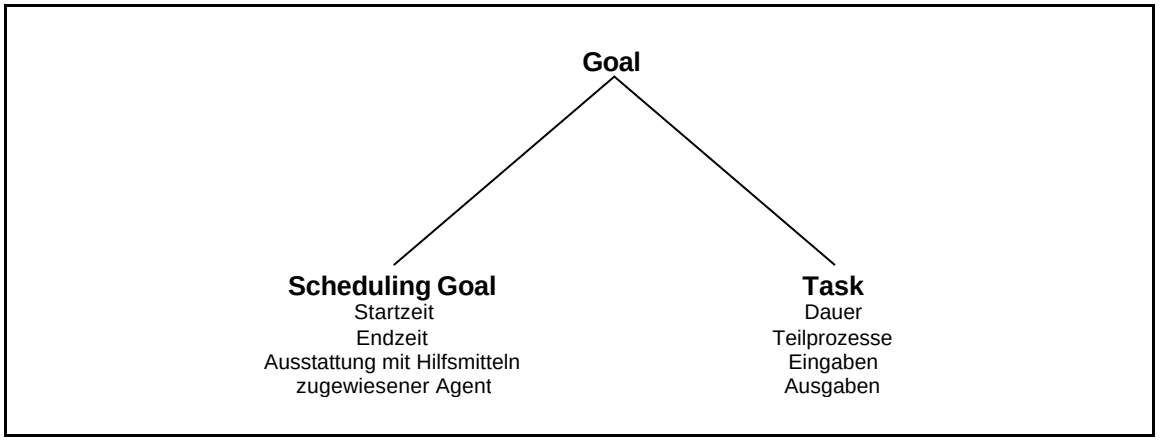


Abbildung 3.31: Beziehung zwischen Goal, Task und Scheduling Goal

Durch die Festlegung welche Eingaben beim Start einer Tasks bereit sein müssen und das Ermitteln derer, die die dazu passenden Ausgaben liefern, entstehen die vorrangigen Beziehungen zwischen den Tasks. Diese Beziehungen ergeben sich aus der Notwendigkeit, daß eine Task, die gewisse Ausgaben produziert, vor einer Task ausgeführt werden muß, die diese als Eingaben benötigt. Die dadurch entstehende sequentielle Ordnung, noch ohne die Zuweisung von Start- und Endzeitpunkten, ist der Ablaufplan.

Tasks werden von Agenten mittels verteilter Ressourcen ausgeführt. Ein **Agent** ist ein menschlicher Bearbeiter oder ein Computerprogramm, der über ein Interface auf Procura zugreift. Dieses Interface muß den Agenten mittels eines Benachrichtigungssystems immer auf dem Neuesten halten, wenn sich für ihn und seine Arbeit durch Planung und Ausführung anderer Prozesse Änderungen ergeben. Die verteilten **Ressourcen** sind in dieser Version von Procura nur Ressourcen, die nicht aufgebraucht werden können. Dies sind im Normalfall Arbeiter und die Arbeitsausstattung, wie Maschinen und Material. Sie haben keine Einsicht in den Arbeitsablauf.

Agenten und Ressourcen müssen bei der Zeitplanung den verschiedenen Tasks für eine bestimmte Zeitdauer zugewiesen werden. Jede Task bekommt dabei einen festen Prozentsatz der verfügbaren Zeit des Agenten oder der Ressourcen zugesprochen. Die Summe aller Tasks, die ein Agent während einer bestimmten Zeit gleichzeitig bearbeitet oder die eine Ressource zur

Verfügung stehen muß, entsprechen dem Nutzungsgrad. Ist ein Agent oder eine Ressource verfügbar, so bewegt sich deren Nutzungsgrad zwischen 0% und 100%. Bei Nicht-Verfügbarkeit muß der Nutzungsgrad 0% betragen.

Durch die Zuweisung eines Agenten und der benötigten Ressourcen zu jeder Task eines Ablaufplans, können die Start- und Endzeitpunkte aller Tasks ermittelt und der vollständige Zeitplan erstellt werden. Dieser Zeitplan enthält den kritischen Pfad und für jede Task ihren frühestmöglichen und spätestmöglichen Start- und Endzeitpunkt.

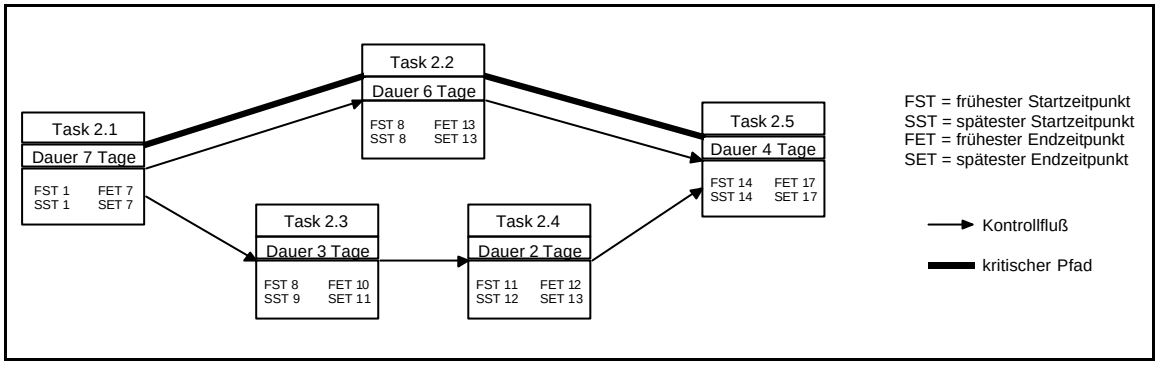


Abbildung 3.32: Beispiel für einen Zeitplan

Wie in Abbildung 3.32 zu sehen, befindet sich der kritische Pfad zwischen den Tasks 2.1, 2.2 und 2.3. Hier dürfen keine Zeitüberschreitungen auftreten. Die Start- und Endzeitpunkte müssen eingehalten werden, um den Zeitplan einzuhalten. Die Tasks 2.3 und 2.4 sind dagegen unkritisch. Bei ihnen kann sich der Start- und Endzeitpunkt, aufgrund eines frei verfügbaren Tages, jeweils um einen Tag verschieben.

Procura erlaubt die hierarchische Anordnung (Baumstruktur) eines Aufgabenplans (Abbildung 3.33). Dabei kann eine Task (Vater) in Sub-Tasks (Söhne) zerlegt werden, die Teilgebiete der Vater-Task behandeln. Für jede Sub-Task wird dann wieder eine eigene Planung durchgeführt. Dies geschieht immer von oben nach unten, also vom Vater zum Sohn (top-down).

Während die Aufgaben- und Zeitplanung im Baum weiter nach unten schreitet, werden immer mehr detaillierte Informationen über Dauer, Zeitpunkte, Eingaben, Ausgaben und benötigte Ressourcen bekannt. Dadurch wird es unter Umständen nötig, die Planungsentscheidungen in höheren Regionen des Baums zu korrigieren und damit die Pläne zu ändern.

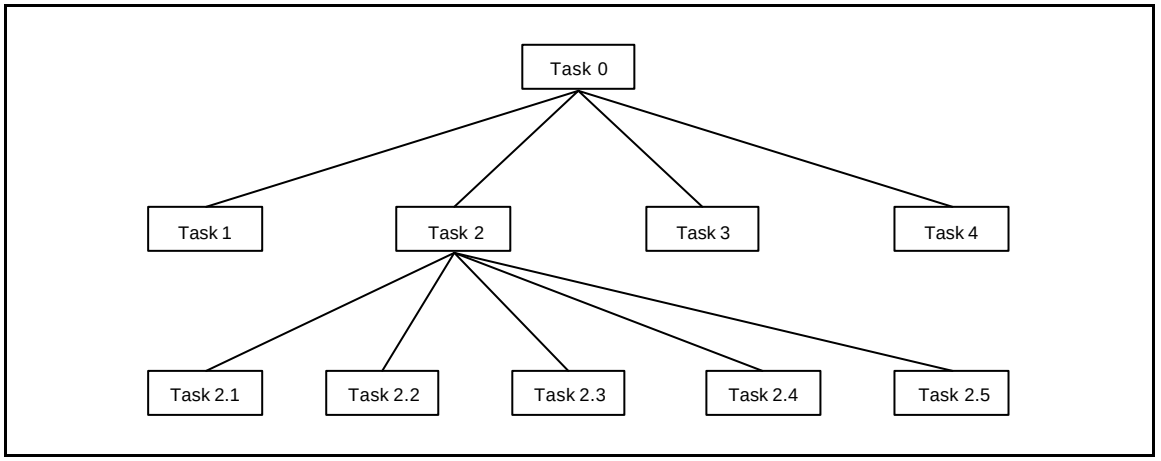


Abbildung 3.33: Beispiel für einen hierarchischen Aufgabenplan

Procura erlaubt es Agenten, Tasks neue Eingaben zuzuweisen und Eingaben, die als notwendig deklariert waren, abzulehnen. Dies ist der Fall, wenn der Agent erkennt, daß die Eingabe für die Erfüllung der Aufgabe nicht erforderlich ist. Erkennt ein Agent, daß er mit dem aktuellen Wert einer Eingabe nicht weiterarbeiten kann, so muß dies manuell, also außerhalb des Systems geregelt werden. Im Falle einer Änderung bei den Eingaben muß der vorab festgelegte Plan an die neue Situation angepaßt beziehungsweise korrigiert und alle betroffenen Agenten davon in Kenntnis gesetzt werden.

Während der Planung und Ausführung einer Task können natürlich auch Konflikte auftreten. Beispielweise kann ein Agent dazu eingeteilt werden mehr als 100% seiner verfügbaren Zeit in verschiedene Tasks einzubringen. Oder zwei Agenten machen Zuweisungen an Variablen die sich gegenseitig ausschließen. Der Agent, der einen solchen Konflikt entdeckt, macht ihn dem System bekannt, das dann seinerseits alle anderen betroffenen Agenten benachrichtigt. Diese Agenten müssen nun eine Lösung für den Konflikt finden und eine oder mehrere Entscheidungen, die zum Konflikt geführt haben, ablehnen. Der Konflikt wird als Rechtfertigung für die Ablehnung ‘aufbewahrt’, so daß zu einem späteren Zeitpunkt, wenn der Konflikt auf andere Art und Weise gelöst wird, wieder zur ursprünglichen Entscheidung zurückgekehrt werden kann. Die Agenten werden auch davon entsprechend in Kenntnis gesetzt.

Das System macht ebenso Aufzeichnungen von abgeschlossenen (committed) Tasks. Dabei gibt es drei Ebenen des Commitments:

1) Eine Task wurde beendet, die ihren Ausgabeparametern Werte zugewiesen hat. Die Entscheidungen können ohne weiteren Aufwand an Zeit und Ressourcen zurückgenommen werden, indem die Zuweisungen ungültig gemacht werden.
   Beispiel: In der Task wurde entschieden, Motor A für ein neues Fahrzeug zu verwenden. Die Entscheidung kann später in Motor B revidiert werden, indem einfach die Ausgabeparameter überschrieben werden.

2) Eine Task wurde beendet, die Task-übergreifende Änderungen durchgeführt hat. Diese Entscheidungen können ebenfalls rückgängig gemacht werden. Um jedoch wieder zum Originalzustand zurückzukehren, ist ein zusätzlicher Aufwand an Zeit und/oder Ressourcen notwendig.
   Beispiel: Nach der Entscheidung für Motor A, wurden bereits Maschinen zu deren Herstellung installiert. Bei einer Änderung der Entscheidung auf Motor B, müssen diese Maschinen mit Zeit- und Materialaufwand erst wieder deinstalliert werden, bevor die Maschinen für Motor B aufgebaut werden können.

3) Während der Ausführung einer Task wird etwas Task-übergreifendes geändert, das nicht rückgängig gemacht werden kann. Dieser Fall tritt beispielsweise auf, wenn nur ein Objekt eines bestimmten Typs existiert, das gelöscht oder irreversibel verändert wurde.

Wenn ein Benutzer versucht Entscheidungen zurückzunehmen, die zur Ablehnung einer abgeschlossenen Task führen, generiert das System eine Warnmeldung.

3.4.2 Architektur

Die Procura-Architektur wurde als Umgebung für den verteilten Einsatz von Ingenieurs-Anwendungen entwickelt. Sie repräsentiert ein Agenten-basiertes System und besteht derzeit aus vier Komponenten: Project-Manager, Agent-Manager, Redux’ und den Application Agents. Jede Komponente ist als einer oder mehrere unabhängige Agenten implementiert. Der Project-

Manager und der Agent-Manager bilden dabei den Kern des Procura-Modells. Abbildung 3.34 zeigt die vier Komponenten, die Pfeile dazwischen repräsentieren den Nachrichtenfluß.

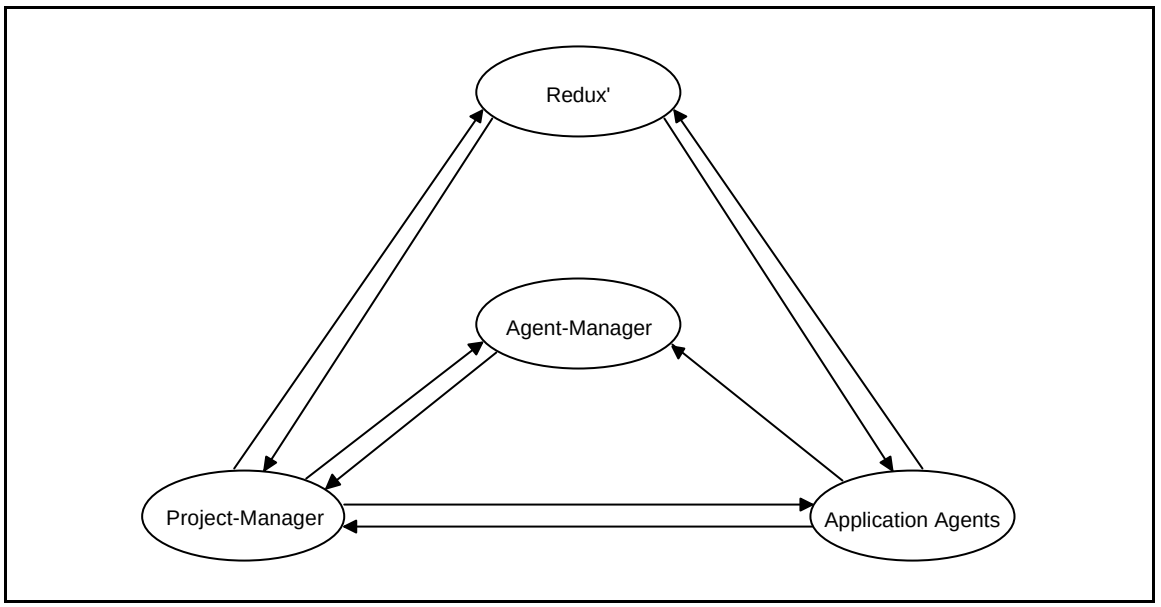


Abbildung 3.34: Procura System-Architektur

Der **Project-Manager** (PM) ist verantwortlich für die Erstellung des Aufgaben- und Zeitplans entsprechend der hierarchischen Struktur und den Beziehungen zwischen den Eingaben und Ausgaben der Tasks. Er hat eine Benutzeroberfläche, die die Pläne oder Teile davon darstellen kann.

Für die Zeitpläne aller Tasks benötigt der PM Informationen über die Verfügbarkeit von Agenten und Ressourcen sowie über deren Fähigkeiten. Diese Informationen erfragt er vom Agent-Manager. Danach erstellt er den Zeitplan, errechnet den kritischen Pfad und für jede Task die frühestmöglichen und spätestmöglichen Start- und Endzeitpunkte.

Während die Agenten Eingaben zu den Tasks, an denen sie arbeiten anfordern oder ablehnen, muß der PM Änderungen an den Plänen einarbeiten und die betroffenen Agenten benachrichtigen. Er ist außerdem für die notwendigen Änderungen an höheren Ebenen der Task-Hierarchie zuständig, die erforderlich werden, wenn die Pläne in den unteren Tasks detaillierter werden. Weiterhin muß er bei entsprechenden Änderungen die Informationen an die Redux'-Komponente weiterleiten.

In der Ausführung eines Plans überwacht der PM die Start- und Endzeiten aller Tasks. Wird eine solche Zeitangabe verletzt, schickt er eine Warnung an die dem Task zugewiesenen Agenten und an die Agenten, die für den übergeordneten Super-Task verantwortlich sind.

Der **Agent-Manager** (AM) kontrolliert den Nutzungsgrad und die Verfügbarkeit aller Agenten und Ressourcen. Für jeden Agenten, Arbeiter und jede Ressource wird dazu ein Terminplan vorgehalten. Meldet sich ein Agent beim System an, so schickt er eine Nachricht mit seinem Namen, seiner Position und seinen Fähigkeiten an den AM. Diese Informationen werden vom AM auf Anfrage des PM oder anderer Agenten weitergegeben.

Wenn ein Agent oder eine Ressource plötzlich ausfällt, so informiert der AM den PM darüber, der dann eine Neuplanung anstößt und die betroffenen Agenten informiert. Dasselbe geschieht, wenn ein Agent, der eigentlich als verfügbar gemeldet ist, zu einem bestimmten Zeitpunkt nicht angesprochen werden kann.

Die dritte Komponente, **Redux'**, unterstützt den Benutzer bei häufiger Neuplanung durch das Sichern der Planungsentscheidungen und der Gründe, die dazu geführt haben, sowie durch das Bestimmen der Konsequenzen von vorgeschlagenen Änderungen. Redux' kann definiert werden als Konflikt-Management-Modell, es stellt dem System den Benachrichtigungs-mechanismus für Konflikte, Entscheidungsablehnungen und der notwendigen neuen Evaluierung von abgelehnten Entscheidungen zur Verfügung (vgl. [Petr93]).

Redux' verteilt Benachrichtigungen an die abhängigen Agenten, wenn

- eine neue Task-Zuweisung erfolgt,
- die Eingaben für eine Task der Agenten verfügbar werden oder sich ändern,
- eine Task für ungültig erklärt wird oder
- die Start- und Endzeiten einer Task sich ändern.

Redux' ist ebenfalls zuständig für die Aufbewahrung einer Liste der bereits abgeschlossenen Tasks. Wird eine Entscheidung abgelehnt, die zur Invalidierung einer abgeschlossenen Task führen würde, so verschickt Redux' entsprechende Warnmeldungen.

Die **Application Agents** sind Agenten, die die Tasks ausführen sollen. Während der Ausführung eines solchen Prozesses führen sie Planungsentscheidungen durch, die Tasks in Sub-Tasks zerlegen und treffen Entscheidungen für die Zuweisungen an die Ausgabeparameter. Sie legen ebenfalls die Commitment-Ebene für die bearbeiteten Tasks fest.

Die Agenten müssen außerdem für jeden erstellten Sub-Task, auf Nachfrage des Project-Manager, die vorgeschlagenen Eingaben und Ausgaben mitteilen. Ebenso die erforderlichen Fähigkeiten des Agenten, die Ressourcen und eine Schätzung der Zeitdauer die zur Lösung der Task benötigt werden. Diese Informationen ermöglichen dem PM eine Zeitplanung für die neu erstellte Task.

Alle Komponenten kommunizieren untereinander über ein definiertes **Nachrichtenprotokoll**. Es gibt außer diesen festgelegten Nachrichten keine Kommunikation zwischen den Komponenten. Auch gibt es für eine Komponente keine Möglichkeit Funktionen einer anderen Komponente aufzurufen. Diese Vorgehensweise sorgt für ein einfaches Austauschen der Komponenten beziehungsweise Agenten.

3.4.3 Beispiel in Procura

Jetzt wollen wir unser Beispiel des Konstruktionsprozesses in der Planung des Procura-Systems betrachten. Dabei muß mit einbezogen werden, daß das Procura-Modell nur Aussagen über die Planung macht und die Koordination der Prozeßabläufe unbeachtet läßt. Wir werden das Beispiel also auch hauptsächlich von der Planungsseite aus betrachten.

Zuerst wird der hierarchische Aufgabenplan erstellt. Die grundlegende Task für das Beispiel sei T0: 'Konstruktion'. Wenn wir annehmen, daß vom Bearbeiter Konstruktionsleiter schon gewisse Planungsentscheidungen getroffen wurden, so könnte die erste Version des hierarchischen Aufgabenplans in etwa wie in Abbildung 3.35 aussehen. Die Konstruktionsaufgabe wurde hier in mehrere Teilaufgaben, T1 bis T4, zerlegt.

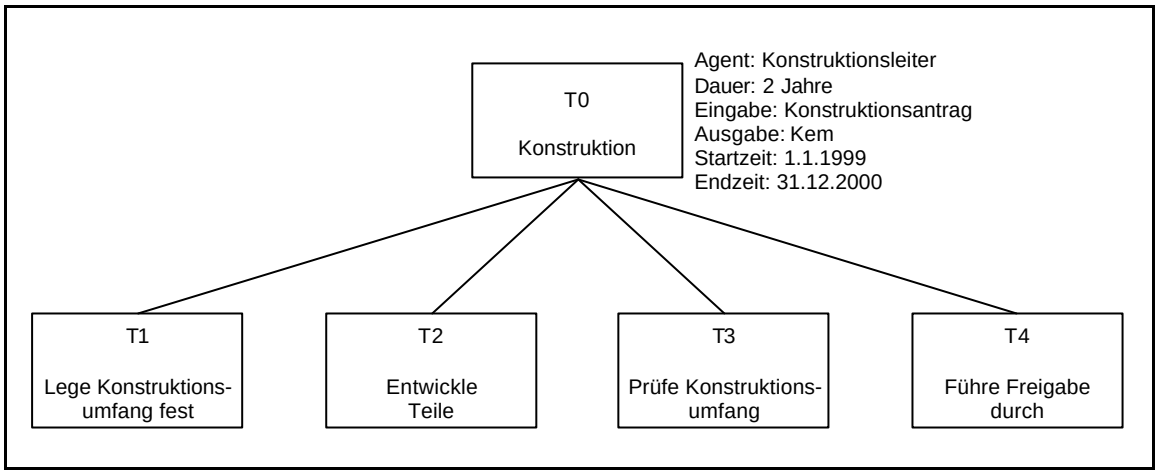


Abbildung 3.35: Erste Version des Aufgabenplans

Nachdem dieser Aufgabenplan dem Project-Manager von Procura bekannt gemacht wurde, benötigt dieser weitere Informationen über die neuen Sub-Tasks. Diese erfragt er von dem Agenten, der die Planungsentscheidungen getroffen hat, hier also vom Konstruktionsleiter. Die vom Agent für jede Sub-Task zu liefernden Informationen umfassen die benötigten Eingaben, die erzeugten Ausgaben, die erforderlichen Anforderungen an den zuständigen Agenten, die nötigen Hilfsmittel (Ressourcen) für die Bearbeitung und die Zeitdauer, die für die Bearbeitung vorgesehen ist.

Sobald der Project-Manager die erforderlichen Informationen besitzt, kann er den Ablaufplan ermitteln. Dabei ordnet er die Tasks entsprechend der Beziehungen ihrer Eingaben und Ausgaben. Bei unserem Beispiel gibt es dabei bis zu diesem Zeitpunkt keine Verzweigungen. Der Kontrollfluß ordnet die Tasks durchgehend von T1 bis T4 (siehe Abbildung 3.36).

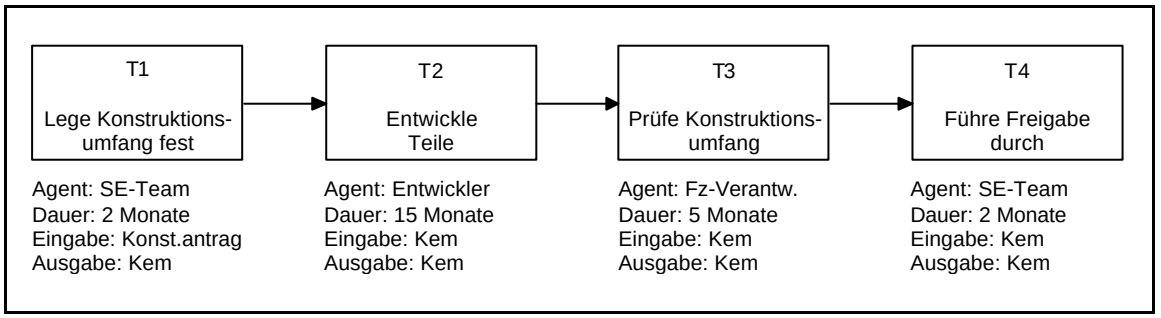


Abbildung 3.36: Erste Version des Ablaufplans

Anschließend erfragt der Project-Manager vom Agent-Manager die für die Tasks in Frage kommenden Agenten. Die Informationen des Agent-Manager zu jedem Agenten enthalten die Zeiten, zu denen dieser zur Verfügung steht. Jetzt kann der Project-Manager passende Agenten den Tasks zuweisen und damit die frühesten und spätesten Start- und Endzeitpunkte errechnen. Mit den vollständigen Zeitangaben läßt sich der Zeitplan und der kritische Pfad festlegen. In unserem Beispiel entspricht er im initialen Zeitplan natürlich genau dem Kontrollfluß, da es keine parallelen Zweige gibt (Abbildung 3.37).

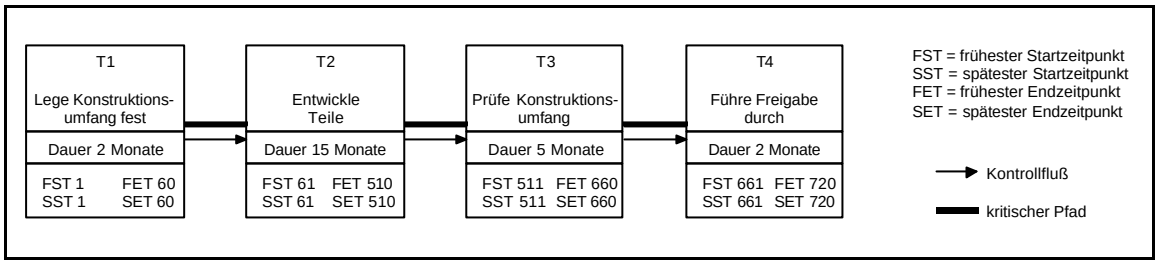


Abbildung 3.37: Erste Version des Zeitplans

Jetzt kann mit der Ausführung des Konstruktionsprozesses begonnen werden. Nach dem Festlegen des Konstruktionsumfangs wird die Entwicklung der zu ändernden Teile gestartet. Dort muß der zuständige Agent zuerst seinen Prozeßablauf für Task 2 (‘Entwickle Teile’) planen. Eine Möglichkeit dies zu tun, wäre die Aufteilung der gesamten Entwicklung anhand der hierarchischen Struktur eines Kem-Objekts. Dabei kann die Entwicklung in die verschiedenen DMUs (‘Entwickle Dmu x’) zerlegt werden. DMUs lassen sich weiter in einzelne Module, und diese wiederum in einzelne Bauteile teilen. Abbildung 3.38 zeigt den Aufgabenplan nach zwei Planungsentscheidungen auf den Tasks ‘Entwickle Teile’ und ‘Entwickle Dmu 2’.

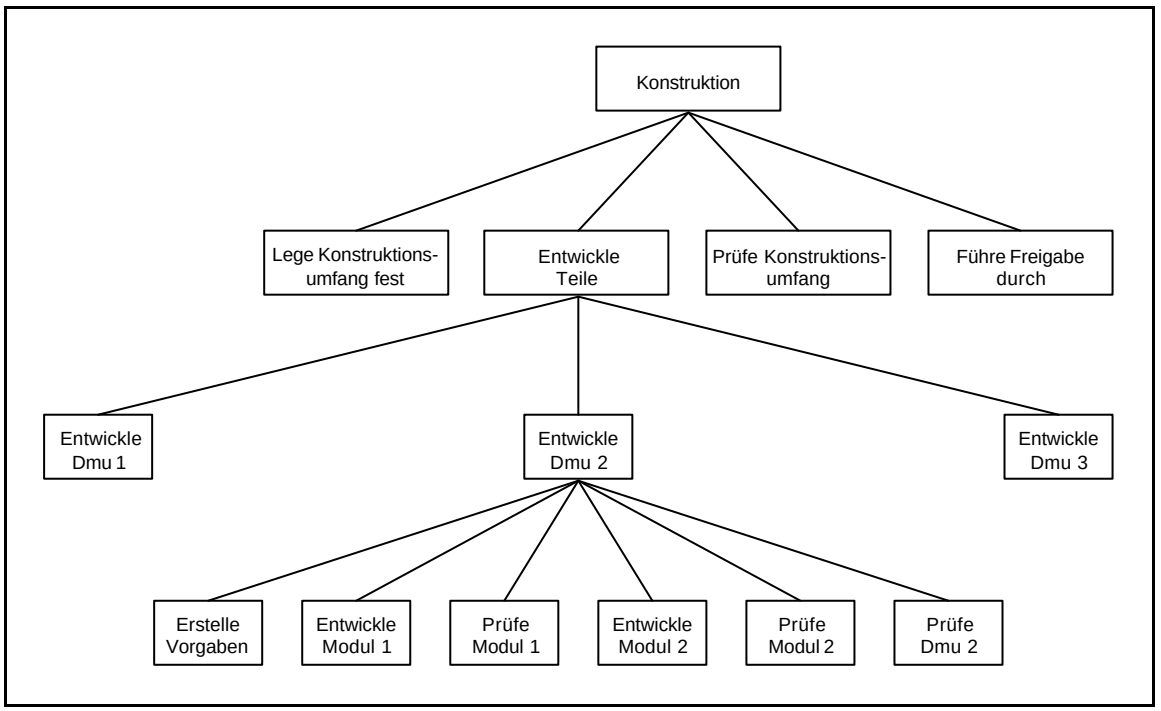


Abbildung 3.38: Aufgabenplan nach weiteren Planungsentscheidungen

Ein Zeitplan für die Task ‘Entwickle Dmu 2’ könnte dann wie in Abbildung 3.39 aussehen. Die Tasks ‘Entwickle Modul 1’ und ‘Prüfe Modul 1’ können parallel zu den Tasks ‘Entwickle Modul 2’ und ‘Prüfe Modul 2’ ausgeführt werden. Danach werden die zwei parallelen Prozeßflüsse in der Task ‘Prüfe Dmu 2’ wieder zusammengeführt. Der kritische Pfad verläuft, wie anhand der Start- und Endzeitpunkte deutlich zu sehen ist, über den Prozeßfluß von Modul 1. Die Entwicklung und Prüfung von Modul 2 beansprucht weniger Zeit. Dadurch können die wirklichen Start- und Endzeitpunkte der Tasks von Modul 2 etwas flexibler ausgewählt werden.

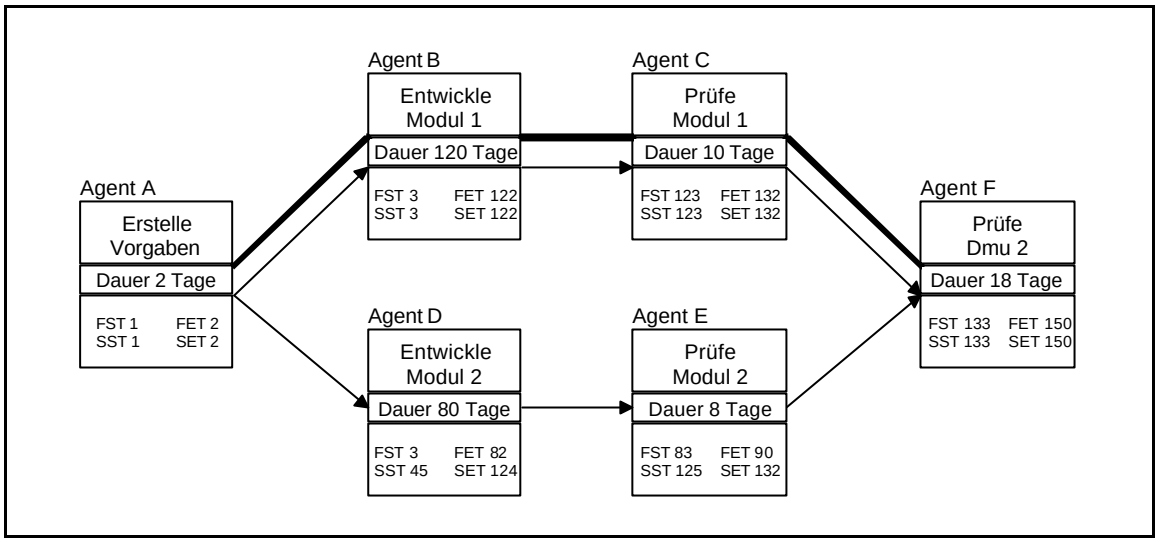


Abbildung 3.39: Zeitplan unterhalb der Task ‘Entwickle Dmu 2’

Wenn jetzt Agent B kurzfristig ausfällt, beispielsweise wegen Krankheit, muß das System darauf reagieren. Dies geschieht, indem Agent B seinen Ausfall beim Agent-Manager bekannt macht. Der Agent-Manager benachrichtigt den Project-Manager, der daraufhin seine Zeitplanung für die Task ‘Entwickle Modul 1’ zurücknimmt. Jetzt fordert der Project-Manager beim Agent-Manager Informationen über einen neuen Agenten mit den passenden Fähigkeiten an. Wenn ein passender Agent (Agent X) existiert, wird überprüft, wann er für die Aufgabe zur Verfügung steht. Angenommen Agent X hat erst ab Tag 4 Zeit, so ist damit der kritische Pfad der Zeitplanung verletzt und dadurch verschiebt sich der gesamte Zeitplan um einen Tag nach hinten. Der Project-Manager ändert nun den Zeitplan dahingehend und informiert alle betroffenen Agenten darüber. Betroffene Agenten sind die, die an Tasks arbeiten, die auf Ausgaben der geänderten Task (hier ‘Entwickle Modul 1’) warten. Im Beispiel also die Agenten C und F.

Sollte Agent B nun plötzlich doch wieder verfügbar werden, teilt der Agent-Manager dies Redux’ mit. Redux’ informiert den Project-Manager, daß die Möglichkeit besteht zur alten Entscheidung, die Agent B der Task ‘Entwickle Modul 1’ zugeordnet hat, zurückzukehren. Der Project-Manager kann dann entscheiden, welche Entscheidung, das heißt, ob Agent B oder Agent X besser geeignet ist. Der Entschluß des Project-Managers kann von vielen Faktoren abhängen. So kann es besser sein, Agent B zu wählen, da dann der Prozeß einen Tag früher beendet wird. Im Gegensatz dazu, kann es mehr Aufwand erfordern, den Zeitplan nochmals zu ändern. Insbesondere dann, wenn geänderte Tasks bereits gestartet wurden.

Gesetzt den Fall, die Task ‘Entwickle Modul 1’ wurde bereits von Agent X ausgeführt und der Project-Manager hat sich dennoch für Agent B entscheiden, so wird die Task erneut in den Zeitplan eingefügt und Agent B zugewiesen. Die bereits ausgeführte Version verbleibt mit ihren Ausgabedaten jedoch im Prozeßgraph. So kann, sollten weitere Änderungen nötig werden, doch wieder auf dieses gesicherte Ergebnis zugegriffen werden.

3.4.4 Abgrenzung und Bewertung

Procura läßt sich sehr gut als Projekt-Management-System einordnen. Die Ansatzpunkte des Procura-Modells beschäftigen sich hauptsächlich mit der Planung eines Projekts. Dabei wird das für Projekt-Management-Systeme typische Zeit- und Ressourcenmanagement unterstützt. Die Erweiterungen in Kontrast zu herkömmlichen Systemen beruhen auf der Integration der drei Phasen Aufgabenplanung, Zeitplanung und Planausführung. Die Phasen werden nicht mehr hintereinander abgearbeitet, sondern ineinander verwoben. Dabei werden während der Ausführung erkannte Änderungen wieder direkt über die Planung mit eingebunden. Die Planung kann so als dynamischer Aufbau der Tasks bezeichnet werden.

Ein weiterer Unterschied zu bisherigen Systemen mit der gebräuchlichen Client-/Server-Architektur, ist die agenten-basierte Arbeitsweise von Procura. Diese Arbeitsweise kann als verteiltes System betrachtet werden. Weitere Agenten können einfach ins System eingeklinkt werden, indem sie über ein Modul, das als Schnittstelle fungiert, Zugriff auf das Procura-Nachrichtenprotokoll bekommen. Dabei ist es nebensächlich, ob es sich bei dem Agenten um einen Bearbeiter oder um eine Systemkomponente handelt. Das Procura-System implementiert ein Benachrichtigungssystem, das die von einer Aktion betroffenen Agenten auf dem aktuellen Stand hält. Nicht betroffene Agenten bekommen keine Informationen. Die Agenten werden also vom System informiert, anstatt sich ihre Informationen bei einem Client-/Server-System selbst abzuholen. Dadurch entstehen die bekannten Vor- und Nachteile eines verteilten Systems (vgl. [Tane95], [Dada96], u.a.).

Über die Kontrolle der geplanten Prozesse während der Ausführung werden so gut wie keine Angaben gemacht. So wird offenbar nur überprüft, ob ein Agent (Bearbeiter) eine ihm zugewiesene Task aufgrund der ihm zur Verfügung stehenden Zeit überhaupt starten kann. Eine Kontrolle während der Ausführung einer Task, ob die geplanten Zeiten eingehalten werden können, findet nicht statt. Anpassungen des Plans können nur wieder nach Beendigung der Task erfolgen. Weiter gibt es diesbezüglich auch keine Aussagen, wie das Procura-System auf Änderungen an den Datenobjekten, also an den Eingaben einer Task reagiert. Zwar findet eine Anordnung aller Sub-Tasks einer Task im Ablaufplan anhand der Abhängigkeiten der Eingaben einer Task zu den Ausgaben anderer Tasks statt, aber ob Änderungen diesbezüglich später wieder in die Planung miteinbezogen werden können, ist nicht bekannt. Diese fehlende Unterstützung verbietet von vornherein weiterführende Mechanismen des Simultaneous Engineering, wie zum Beispiel die vorzeitige Datenweitergabe.

Ein weiterer Kritikpunkt im Hinblick auf Produktentwicklungsprozesse ist die Art und Weise, wie Procura strukturierte Arbeitsabläufe verwendet. Zu Anfang einer Prozeßplanung betrachtet Procura immer nur die Aufgabenstruktur, den Aufgabenplan. Ein strukturierter Arbeitsablauf wird nur innerhalb einer Task, mit deren Sub-Tasks, unterstützt. Dieser strukturierte Arbeitsablauf kann jedoch nicht entsprechend einem Kontrollfluß geregelt werden, er entsteht aus den Abhängigkeiten zwischen Eingaben und Ausgaben der enthaltenen Tasks. Wichtig ist außerdem, daß erst dann wieder auf die Super-Task zugegriffen werden kann, wenn der Arbeitsablauf der Sub-Tasks beendet ist. Damit schafft Procura künstliche Grenzen, über die eine eventuelle Datenweitergabe nicht möglich ist, da sie nicht modelliert werden kann. Für Produktentwicklungsprozesse ist dieses Vorgehen weniger geeignet. Sie sind für gewöhnlich bis zu einem bestimmten Detaillierungsgrad vorstrukturierte Prozesse und brauchen im Normalfall nur einmal geplant zu werden. Dann muß klar sein, was produziert oder geändert werden muß, und der Prozeß wird immer entsprechend der Vorstrukturierung abgearbeitet. Weiterhin ist vorzeitige Datenweitergabe nur möglich, wenn ein Arbeitsablauf von Anfang bis Ende in einer Struktur modelliert werden kann.

Auch für Gruppenarbeitsmechanismen ist bei Procura keine Unterstützung vorgesehen. Wollen sich mehrere Agenten gemeinsam über das weitere Vorgehen oder über Änderungen unterhalten, so muß dies manuell mittels externer Anwendungen geregelt werden.

3.5 CoMo-Kit

Das CoMo-Kit-Projekt der Universität Kaiserslautern[2] konzentriert sich auf Methoden und Techniken zur Unterstützung der Planung und Koordination komplexer, verteilter Entwicklungs- und Entwurfsprojekte (vgl. [Maur96]). Es wurde hauptsächlich für zwei Anwendungsgebiete konzipiert: Software Engineering und Bebauungsplanung. Die Verfahrensweise von CoMo-Kit erlaubt dabei das Ineinandergreifen von Planung und Ausführung. Dies ist essentiell für Entwurfsprozesse, da sie nur dann entsprechend detailliert geplant werden können, wenn einige Schritte bereits ausgeführt wurden.

Für ein neues Projekt muß in einem ersten Schritt ein initialer Projektplan erstellt werden. Dieser Plan enthält Beschreibungen über die zu erledigenden Aufgaben, Definitionen der Datenstrukturen, die während der Ausführung angelegt werden müssen und eine Liste der Bearbeiter, die in das Projekt eingebunden sind. Der erstellte Projektplan wird dann vom CoMo-Kit-Scheduler, der als Workflow-Management-Server fungiert, verwendet, um die Projektabarbeitung zu unterstützen.

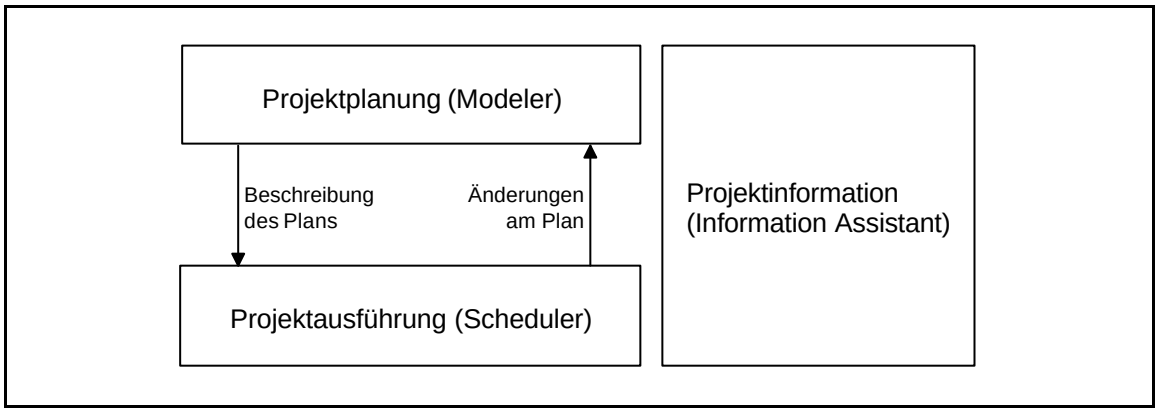


Abbildung 3.40: CoMo-Kit System-Architektur

Abbildung 3.40 zeigt die System-Architektur von CoMo-Kit. Es besteht im wesentlichen aus drei Komponenten:

- Der **Modeler** erstellt den Projektplan.
- Der **Scheduler** unterstützt die Durchführung eines Projekts und verwaltet die erstellten Projektdaten.
- Der **Information Assistant** erlaubt den Zugriff auf den aktuellen Zustand eines Projekts.

Im folgenden werden die Konzepte beschrieben, die hinter den ersten zwei Komponenten stehen.

[2] Im Rahmen des Sonderforschungsbereichs 501 der Deutschen Forschungsgemeinschaft

3.5.1 Projektplanung

Projektplanung ist das Erstellen eines Modells, das aussagt, wie das Projekt durchgeführt werden sollte. Bei groß angelegten Projekten kann ein detaillierter Plan nicht vor Beginn der Ausführung entwickelt werden. Beginnend mit einem Anfangsplan können die ersten Aufgaben ausgeführt werden. Danach kann der Plan, auf den Ergebnissen basierend, verbessert beziehungsweise erweitert werden. Dies bedeutet ein Ineinandergreifen von Planung und Abarbeitung eines Projekts.

Das CoMo-Kit-Modell verwendet vier grundlegende Notationen, um kooperative Entwicklungsprozesse zu modellieren: Tasks, Products, Methods und Resources.

Eine **Task** (Aufgabe) beschreibt ein Ziel, das während der Durchführung des Projekts erreicht werden soll. Tasks können in weitere Sub-Tasks unterteilt werden. Die hauptsächlich angestrebten Ziele sind das Erstellen von Informationen. Informationen, die für das Projekt von wesentlicher Wichtigkeit sind, werden im Projektplan beschrieben. Hier kann jedoch nur beschrieben werden, welcher Typ von Informationen benötigt wird oder produziert werden muß.

Jede Task ist mit einem Satz von Eingabe- und Ausgabeparametern (Variablen) verknüpft. Eingabeparameter können definiert werden als für die Planung notwendig, für die Ausführung notwendig und optional. Ausgabeparameter können optional oder erforderlich sein. Zusätzlich kann für jede Task eine Vorbedingung und eine Nachbedingung festgelegt werden. Vorbedingungen können verwendet werden, um die Eingabeparameter auf gewisse Voraussetzungen zu überprüfen. Nachbedingungen werden verwendet, um die Ausgabeparameter auf eine gewünschte Qualität zu testen.

Während der Ausführung der Tasks werden Entscheidungen getroffen, die in der Zuweisung von Werten zu den Ausgabeparametern resultieren. CoMo-Kit nimmt nun an, daß eine kausale Abhängigkeit zwischen den verfügbaren Eingabeparametern und den erstellten Ausgabeparametern besteht. Dabei wird prinzipiell davon ausgegangen, daß in der Planung einer Task nur Eingabeparameter zugewiesen werden, die für die Erreichung des Ziels relevant sind.

Products sind Objekte, die im Verlauf der Projektausführung erstellt werden. Um sie zu modellieren, wird ein objekt-orientierter Ansatz benutzt. Dabei wird, wie üblich, zwischen Klassen und Instanzen unterschieden. Während der Ausführung werden Produktinstanzen als Werte repräsentiert, die Eingabe- oder Ausgabeparametern zugewiesen werden. Der Typ des Parameters wird durch die Produktklasse spezifiziert.

Um das Ziel einer Task zu erreichen, werden **Methods** (Methoden) angewandt. Für jede Task können mehrere, auch vordefinierte, alternative Methoden existieren. Neue Methoden können definiert werden, wenn eine Task geplant oder der Plan überarbeitet wird. Tasks oder Sub-Tasks können durch Methoden so weiter unterteilt werden. Dabei muß man die Unterteilung einer Task in Sub-Tasks als UND-Verzweigung und die Unterteilung in Methoden als ODER-Verzweigung betrachten. Sub-Tasks sind Teilaufgaben, die alle ausgeführt werden müssen, um das Ziel einer Task zu erreichen. Methoden sind Lösungsmöglichkeiten, die alle zum selben Ergebnis führen und von denen genau eine ausgeführt werden muß, um das Ziel zu erreichen (siehe Abbildung 3.41).

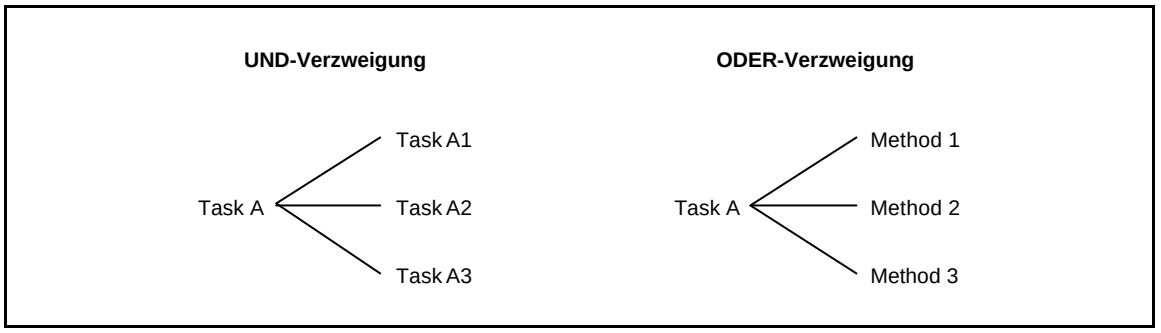


Abbildung 3.41: Tasks und Methods bei CoMo-Kit

Methoden werden von Agenten (Bearbeitern) ausgeführt. Sie treffen die Entscheidung für eine der zur Verfügung stehenden Methoden abhängig vom aktuellen Kontext des Projekts. Unterschieden wird hier zwischen atomaren und komplexen Methoden.

Atomare Methoden weisen Ausgabeparametern Werte (Instanzen einer Produktklasse) zu. Prozeßskripte beschreiben, wie menschliche Bearbeiter diese Task auflösen können und in einer formalen Sprache spezifizierte Programme beschreiben, wie die Task automatisch und ohne Interaktion bearbeitet werden kann.

Komplexe Methoden werden durch einen Flußgraph beschrieben. Der Flußgraph besteht aus Knoten die Tasks, Sub-Tasks und Parameter darstellen, sowie aus Verknüpfungen (Links), die die Beziehungen zwischen Parametern und Tasks definieren. Die Richtung der Verknüpfungen gibt an, ob es sich dabei um Eingabe- oder Ausgabeparameter handelt.

Resources (Ressourcen) für die Projektplanung und die Ausführung der Tasks sind Agenten und Tools. Die aktiven Agenten benutzen dabei die passiven Tools (Hilfsmittel), um eine Task zu bearbeiten. Agenten können menschliche Bearbeiter oder aber auch automatisch agierende Maschinen beziehungsweise Computerprogramme sein.

Für jede Task werden im Projektplan die Anforderungen definiert, die ein Agent erfüllen muß, um die Task zu bearbeiten. Desweiteren speichert das System die Eigenschaften eines jeden Agenten, um so während des Projektablaufs die Agenten zu ermitteln, die für die Lösung einer speziellen Aufgabe in Frage kommen.

3.5.2 Projektausführung

Für die Projektausführung ist der Scheduler zuständig. Für jedes Projekt wird beim CoMo-Kit eine neue Instanz des Schedulers erstellt und mit der Start-Task initialisiert. Bei der Initialisierung wird die Task einer Gruppe von Agenten zugewiesen, die laut Projektplan in der Lage sind, sie zu bearbeiten. Nun wartet der Scheduler so lange, bis ein entsprechender Agent sich anmeldet und die Task annimmt. Als erstes wählt der Agent eine Methode, die die Task in verschiedene detailliertere Sub-Tasks aufteilt. Da dies ein Planungsschritt ist, kann ein Agent die Task nur akzeptieren, wenn alle für die Planung erforderlichen Informationen verfügbar sind. Danach delegiert der Agent jeden Sub-Task zu der Gruppe der dafür verantwortlichen Agenten. Kann eine Task nicht mehr in Sub-Tasks zerlegt werden, ist der Agent selbst für das Erstellen der Ausgabeparameter verantwortlich. Der Scheduler garantiert, daß so eine Task nur dann von einem Agenten gestartet werden kann, wenn alle notwendigen Eingabeparameter vorhanden sind.

Da das CoMo-Kit-Modell inkrementelle Projektplanung unterstützt, kann sich der aktuell gültige Projektplan jederzeit ändern oder durch das Hinzufügen neuer Methoden erweitern. Dies bedingt eine Nebeneinanderausführung der Planung und Bearbeitung der Tasks. Ein kombinierter Planungs- und Ausführungszyklus sieht also in etwa so aus:

- Start-Task definieren oder Task von übergeordneter Stelle annehmen.
- Ziel der Task beschreiben, Eingabe-, Ausgabeparameter und Bedingungen festlegen.
- Task ausführen
  oder
- Task in Sub-Tasks zerlegen und Datenfluß zwischen ihnen definieren.
  - Sub-Tasks an Agenten delegieren.
  - Ausführung der Sub-Tasks überwachen.
  - Auf Änderungen am Projektplan oder an den Eingabeparametern reagieren.

Die Architektur des CoMo-Kit-Schedulers entspricht einer Client-Server-basierten Arbeitsweise (Abbildung 3.42).

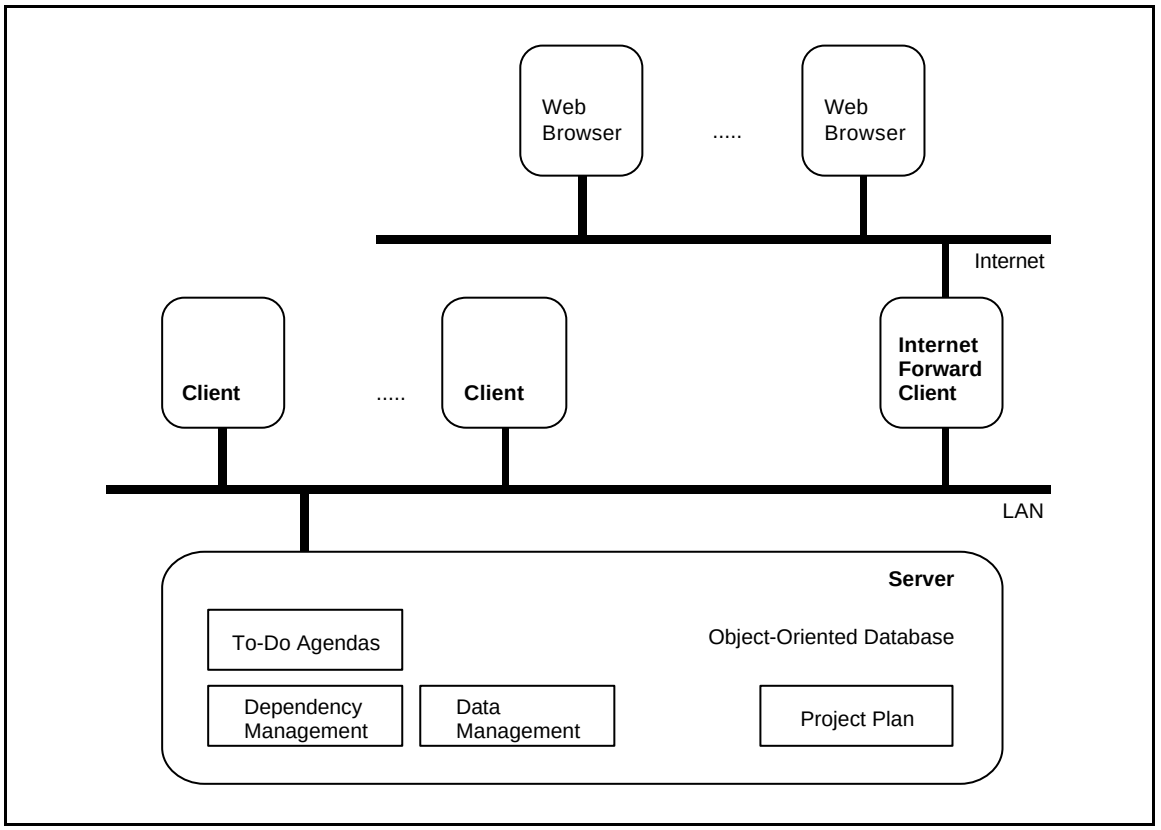


Abbildung 3.42: Architektur des CoMo-Kit Schedulers

Die Server-Komponente des Schedulers basiert auf einer objektorientierten Datenbank[3]. Ihre Aufgaben umfassen das Sichern des aktuellen Projektplans, das Verwalten der Aufgabenliste (**To-Do Agenda**) für jeden Agenten und das Speichern aller während der Ausführung erstellten Objekte (**Data Management**). Weiterhin muß sie die Korrektheit der Abhängigkeiten und

[3] GemStone von Servio Corporation

Beziehungen zwischen den Tasks, sowie der Eingabe- und Ausgabeparameter sicherstellen (**Dependency Management**).

Die Serverkomponente ist von den **Clients** (Klienten) direkt über LAN erreichbar. Die Klienten stellen eine Benutzeroberfläche zur Verfügung die folgende Aufgaben unterstützt:

- Auswählen und Annehmen der Tasks, an denen gearbeitet werden soll,
- Planen einer Task und Ändern eines Projektplans,
- Zerlegen einer Task in Sub-Tasks,
- Überwachen des Arbeitsfortschritts der Sub-Tasks und
- Editieren von Objekten.

Über einen speziellen Klienten[4] (**Internet Forward Client**) ist es möglich, die Klienten-Benutzeroberfläche via Internet auf einem Web-Browser darzustellen. Somit ist es möglich einen Klienten per WWW an den Server anzubinden.

3.5.3 Projektänderungen

Ein Problem, das bestehen bleibt, ist die Unterstützung der Reaktion auf geänderte Entscheidungen, die die Änderung des Projektplans nach sich ziehen. Insbesondere bei einer großen Anzahl von Bearbeitern.

CoMo-Kit verwendet kausale Abhängigkeiten zwischen Entscheidungen, um damit Änderungsprozesse zu unterstützen. Dazu erlaubt es, diese Abhängigkeiten direkt aus dem Projektplan abzuleiten. Die Bearbeiter haben nicht die Pflicht alle Beziehungen zwischen Entscheidungen dem System manuell bekanntzumachen. Eine umfassende Aufstellung der kausalen Abhängigkeiten garantiert die vollständige Nachverfolgung des Entwicklungsprozesses. Kann immer herausgefunden werden, was durch eine Entscheidung beeinflußt wird, so können auch immer alle betroffenen Bearbeiter informiert werden, wenn eine Änderung eintritt. Zur Unterstützung der Nachverfolgung der Abhängigkeiten verwendet CoMo ein *Truth Maintenance System* (TMS) [Doyl79]. Ein TMS ist ein System aus dem Bereich der künstlichen Intelligenz (KI). Es sichert alle getroffenen Entscheidungen und alle daraus gefolgerten Entscheidungen. Somit kann das System immer auf verworfene Entscheidungen, die durch eine Änderung im Projektplan wieder aktuell sind, zurückgreifen, ohne die betroffene Task beziehungsweise Methode erneut zu bearbeiten (vgl. [LuSt93]).

Die Abhängigkeitsverwaltung (Dependency Management) des CoMo-Systems ermittelt aus dem Projektplan immer zwei Arten von Abhängigkeiten:

- Der Datenfluß: Die Ausgabe einer Task ist von der verfügbaren Eingabe abhängig.
- Zerlegung einer Task: Wird eine Task ungültig, so müssen auch alle Sub-Tasks ungültig werden.

Die Abhängigkeiten werden als Gleichungen mit logischen Ausdrücken formuliert. Sie sind generell in der Form: Entscheidung 1 $\Rightarrow$ Entscheidung 2 (aus Entscheidung 1 folgt Entscheidung 2). Drei Beispiele sollen dies nun näher erläutern, die ersten zwei Beispiele zeigen die

[4] Visualwave Package von ParcPlace Systems

Abhängigkeiten des Datenflusses, Beispiel 3 definiert die Abhängigkeiten zwischen Task und Sub-Tasks.

$$decision\left(m_j\right) \Rightarrow assignment\left(O_1 = o_{1j}\right) \wedge \ldots \wedge assignment\left(O_n = o_{nj}\right)$$

Abbildung 3.43: Beispiel 1 einer Abhängigkeitsgleichung

Beispiel 1 (Abbildung 3.43) zeigt die Abhängigkeit der Zuweisungen von Ausgabeparameter von der erfolgreichen Ausführung einer Methode. Hier werden die Ausgabeparameter o_1 bis o_n der Task j den globalen Variablen O_1 bis O_n zugewiesen (assignment). Dies ist von der Ausführung der Methode m der Task j abhängig (decision). Wird das Prädikat der Taskausführung ungültig, so wird durch die hiermit beschriebene Abhängigkeit auch jede Zuweisung der Ausgabeparameter ungültig.

$$\neg assignment\left(I_1 = i_{1k}\right) \vee \ldots \vee \neg assignment\left(I_n = i_{nk}\right) \Rightarrow rejected\_decision\left(m_j\right)$$

Abbildung 3.44: Beispiel 2 einer Abhängigkeitsgleichung

Beispiel 2 (Abbildung 3.44) beschreibt die Gültigkeit der von Eingabeparametern abhängigen Ausgabeparameter. Erfolgt keine Zuweisung an die Eingabeparameter, so wird die Task nicht ausgeführt, beziehungsweise deren Ergebnis zurückgewiesen.

$$decision\left(m_i\right) \Rightarrow validSubTask\left(T_1\right) \wedge \ldots \wedge validSubTask\left(T_n\right)$$

Abbildung 3.45: Beispiel 3 einer Abhängigkeitsgleichung

Das dritte Beispiel (Abbildung 3.45) zeigt, wie nach der Zerlegung einer Task in mehrere Sub-Tasks die Abhängigkeiten beschrieben werden. Die Sub-Tasks T_1 bis T_n sind gültig und müssen ausgeführt werden, solange die Methode m der Task i innerhalb des Projektplans gültig ist und ausgeführt werden muß. Die hier verwendete Methode ist die Standardmethode, die das Aufteilen einer Task in Sub-Tasks erlaubt. Weitere Erklärungen zu diesen Abhängigkeitsgleichungen finden sich in [Maur96].

Anhand dieser Folge von Gleichungen, die das gesamte Projekt repräsentieren, läßt sich nun die Gültigkeit von früher getroffenen Entscheidungen kontrollieren. Wird im Laufe der Abarbeitung eine Entscheidung ungültig, so müssen auch alle davon abhängenden Entscheidungen ungültig gemacht werden. Dadurch können bei Änderungen automatisch die ebenfalls zu ändernden Pläne beziehungsweise zu benachrichtigenden Bearbeiter ermittelt werden.

Eine Grenze für die automatische Ermittlung der kausalen Abhängigkeiten sind die in der Praxis häufig vorkommenden unvollständigen Projektpläne. Dies hat zwei Gründe:

a) Ein Standardplan (beispielweise bezüglich des Qualitätsmanagements ISO9000ff) stimmt nicht exakt mit den wirklichen Anforderungen überein oder ist zu grob gegliedert.

b) Sehr große Projekte benötigen meist auch sehr viel Zeit. Am Anfang sind diese Projekte oftmals nur vorskizziert und nur die ersten Schritte sind detailliert ausgearbeitet. Spätere Schritte können nur dann geplant werden, wenn die Ergebnisse voriger Schritte verfügbar werden.

Für den ersten Fall kann man argumentieren, daß ein 'guter' Standardplan diese Unzulänglichkeit nicht besitzt. Der zweite Fall jedoch kann nicht in geeigneter Weise umgangen werden. Diese zwei Problemfälle resultieren in der Möglichkeit, daß nicht alle kausalen Abhängigkeiten zwischen zwei Entscheidungen modelliert werden. Auch ist es möglich, daß der Projektplan Eingabeparameter angibt, die nicht relevant sind. Um dies zu umgehen, erlaubt das System dem Bearbeiter, zusätzlich zu automatischen Ermittlung, einen manuellen Eingriff in die Gleichungen der kausalen Abhängigkeiten.

3.5.4 Beispiel in CoMo-Kit

Wie könnte nun unser Beispiel des Konstruktionsprozesses mit CoMo-Kit modelliert werden? Als erster Schritt muß ein initialer Projektplan erstellt werden (Abbildung 3.46).

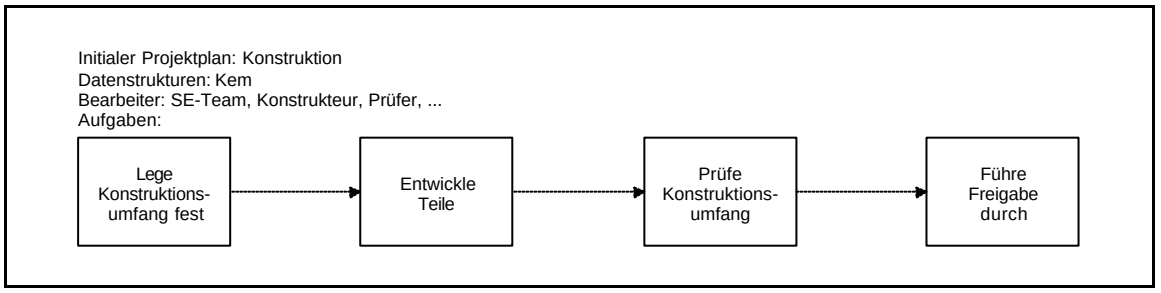


Abbildung 3.46: Der initiale Projektplan

Der initiale Projektplan gibt die Aufgaben vor, die zu erledigen sind, und legt damit einen gewissen Ablauf von vornherein fest. Weiterhin definiert er die benötigten Datenstrukturen und die in den Prozeßablauf involvierten Bearbeiter.

Ist der Plan erstellt, kann die Prozeßausführung mit der Start-Task begonnen werden. Für jede Task wird ihr Ziel, ihre Eingabe- und Ausgabeparameter, der aktuelle Bearbeiter, sowie eventuelle Vorbedingungen und Nachbedingungen beschrieben. Das Ziel der Start-Task, das heißt ihre Aufgabe, ist hier: 'Lege Konstruktionsumfang fest'. Als Eingabeparameter gibt es höchstens eine Konstruktionsanforderung, als Ausgabeparameter wird ein Kem-Objekt festgelegt. Der Bearbeiter ist das SE-Team, als Nachbedingung kann man höchstens festlegen, daß der zu tätigende Konstruktionsumfang im Kem-Objekt festgelegt sein muß.

Die Beziehungen zwischen den Tasks werden durch die Abhängigkeitsgleichungen beschrieben. Aus dem initialen Projektplan abgeleitet, ergeben sich für unser Beispiel nur Abhängigkeiten bezüglich des Datenflusses. Die Eingabe einer Task ist immer von der Ausgabe einer Task, entsprechend der obigen Reihenfolge, abhängig. Für unsere Zwecke wird hier eigentlich immer nur das Kem-Objekt weitergeleitet.

Zur Ausführung einer Task kann ein Bearbeiter mehrere Methoden zur Verfügung haben. Er kann jedoch immer nur eine Methode ausführen. Eine Standard-Methode ist das Aufsplitten der Task in Sub-Tasks - in kleinere Teilaufgaben, die so an andere Bearbeiter weitergegeben

werden können. Ein Beispiel für das Aufsplitten der Task ‘Entwickle Teile’ wird in Abbildung 3.47 verdeutlicht:

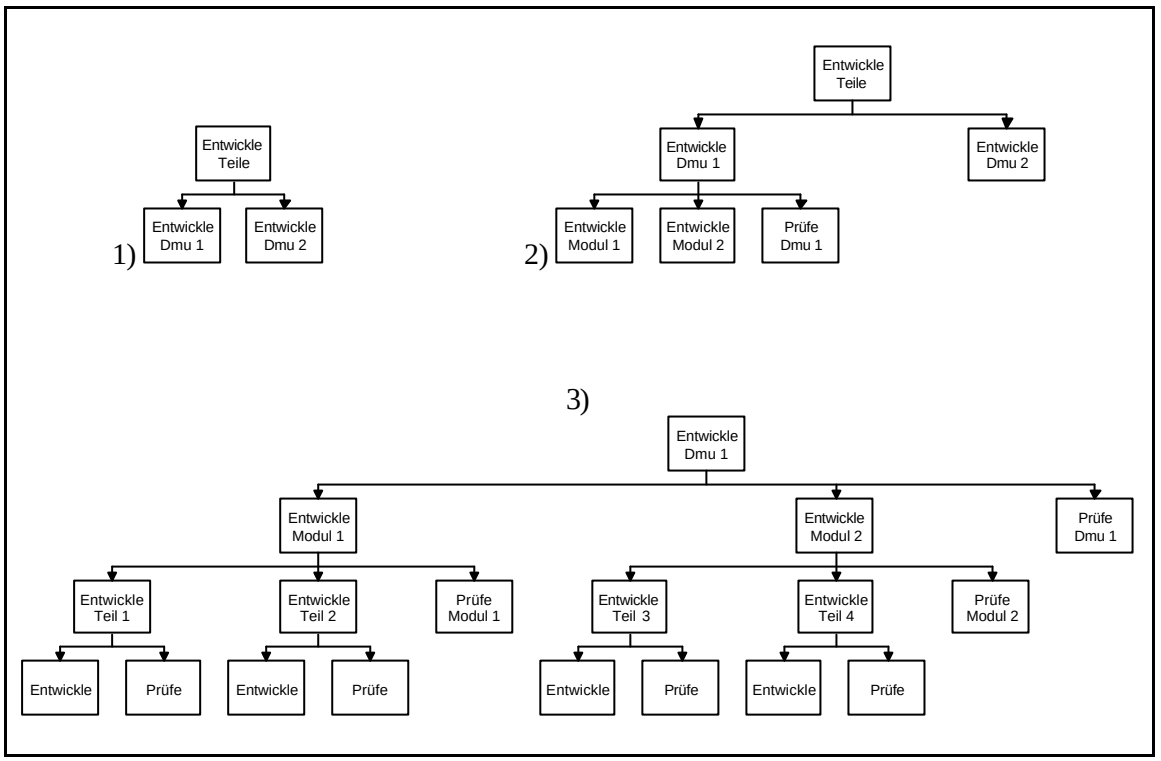


Abbildung 3.47: Entstehung einer Task-Struktur durch Aufsplitten in Sub-Tasks

Darstellung 1 teilt die Task in zwei Sub-Tasks, die jeweils ein DMU der Konstruktion entwickeln. In Darstellung 2 wird eine solche Sub-Task (‘Entwickle Dmu 1’), vom dann zuständigen Bearbeiter wiederum gesplittet. Diesmal in drei Sub-Tasks, wobei zwei je ein Modul entwickeln und eine Task für die Prüfung der DMU zuständig ist. Darstellung 3 schlußendlich zeigt, wie eine vollständige Task-Struktur eines DMU-Zweiges aussehen könnte (dargestellt ohne die initiale Task ‘Entwickle Teile’). Eine Aufsplittung einer Task ist aber wohlgemerkt immer vom Bearbeiter der Task abhängig. Beispielsweise könnte sich der Bearbeiter von Task ‘Entwickle Teil 1’ auch entscheiden die Entwicklung und Prüfung komplett selber durchzuführen, sofern eine passende Methode innerhalb der Task existiert.

Auch beim Aufsplitten einer Task werden Abhängigkeitsgleichungen erstellt. Diese repräsentieren, zu welcher Task welche Sub-Tasks gehören. Durch dieses System der Abhängigkeiten entsteht im Fehlerfall, in Zusammenspiel mit dem Truth Maintenance System, ein Backtracking-Mechanismus, der im folgenden näher beschrieben wird.

Wir betrachten dabei einen Teil der in Abbildung 3.47 dargestellten Task-Struktur, ausgehend von der Task ‘Entwickle Modul 1’. Der Sachverhalt ist in Abbildung 3.48 dargestellt. Wir gehen davon aus, daß die Task ‘Entwickle Teil 1’ mit ihren Sub-Tasks bereits erfolgreich abgeschlossen ist. Ebenso die Sub-Task ‘Entwickle’ von ‘Entwickle Teil 2’. Dies wird durch das O.K.-Symbol dargestellt. Dagegen tritt bei der Sub-Task ‘Prüfe’ ein Fehler auf, angezeigt durch den Blitz. Für den Fehler nehmen wir an, daß bei ‘Entwickle Modul 1’ der Bauraum für Teil 2 zu klein gewählt wurde, und dies erst nach der Entwicklung des Teils, nämlich bei der Prüfung auffällt.

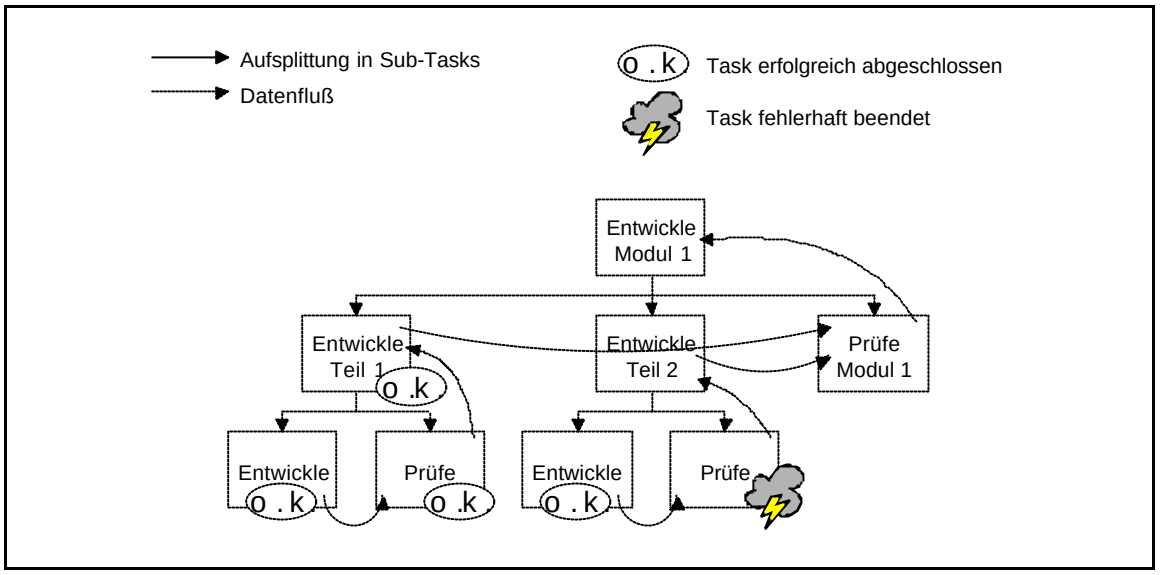


Abbildung 3.48: Beschreibung Backtracking - Bild 1

Die gestrichelten Pfeile zeigen den durch Abhängigkeitsbeziehungen modellierten Datenfluß zwischen den Tasks. Grau dargestellte Tasks sind nicht mehr gültig. Das heißt, sie wurden entweder vor der Ausführung wieder zurückgenommen oder ergebnislos abgebrochen. Wurden sie zuvor bereits erfolgreich beendet, dies wird durch das dann ebenfalls graue O.K.-Symbol angezeigt, wurde ihr Ergebnis vom Truth Maintenance System zwar gespeichert, gilt jedoch im Augenblick als ungültig.

Nachdem der Fehler festgestellt wurde, wird die Datenflußbeziehung zur Super-Task ‘Entwickle Teil 2’ ungültig. Der Bearbeiter der Task kann den Fehler nicht beheben, damit wird die Super-Task selber ungültig. Durch die Abhängigkeitsbeziehungen des Aufsplittens werden beide Sub-Tasks ungültig. Das Truth Maintenance System merkt sich jedoch das Ergebnis der bereits erfolgreich abgeschlossenen Task ‘Entwickle’. Durch die nun ebenfalls ungültig gewordenen Datenfluß-Abhängigkeitsbeziehungen zwischen den Ausgabeparametern von ‘Entwickle Teil 2’ und den Eingabeparametern von ‘Prüfe Modul 1’ wird die letztgenannte Task auch ungültig (siehe Abbildung 3.49).

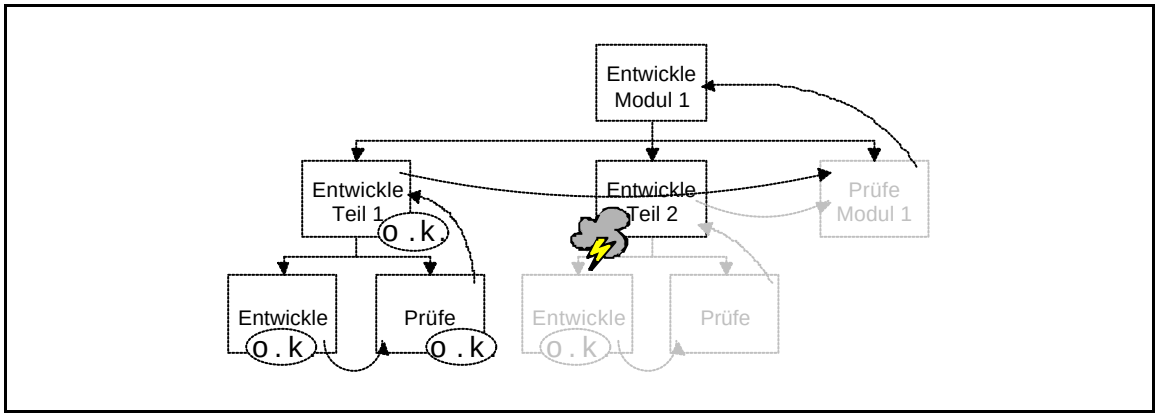


Abbildung 3.49: Beschreibung Backtracking - Bild 2

Der Fehler wird nun, wiederum mittels der Datenflußbeziehung, an die übergeordnete Task ‘Entwickle Modul 1’ weitergegeben. Daraufhin werden auch dort die Abhängigkeits-beziehungen des Aufsplittens ungültig und machen alle Sub-Tasks ungültig (Abbildung 3.50).

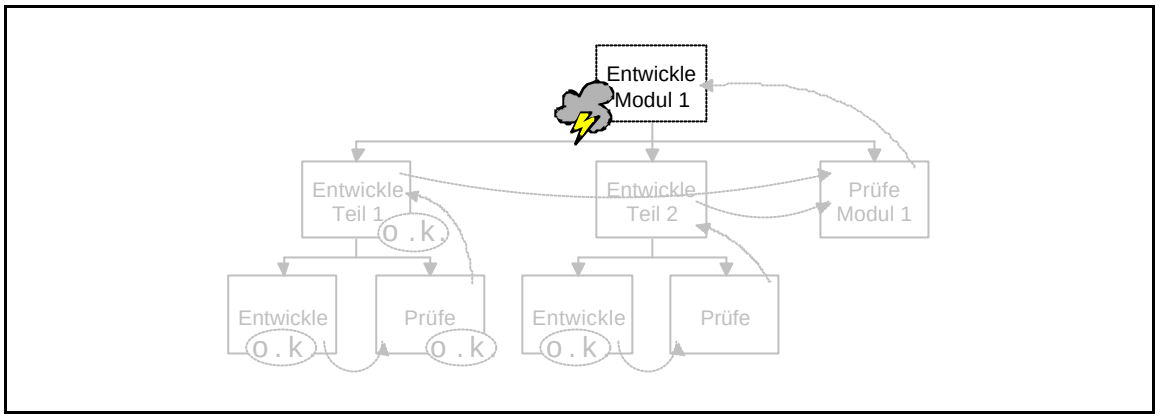


Abbildung 3.50: Beschreibung Backtracking - Bild 3

Jetzt kann der Bearbeiter von Task ‘Entwickle Modul 1’, zum Beispiel der Modul-Verantwortliche, den Fehler bereinigen. Wir gehen der Einfachheit halber davon aus, daß es reicht, den Bauraum für Teil 2 zu vergrößern. Alle Vorgabedaten für Teil 1 können unverändert bleiben.

Nach der Korrektur können alle drei Sub-Tasks wieder neu gestartet werden. Bei der Task ‘Prüfe Modul 1’ muß jedoch noch auf die Ergebnisse der Tasks ‘Entwickle Teil 1’ und ‘Entwickle Teil 2’ gewartet werden. Um zu vermeiden, daß bereits ausgeführte und erfolgreich beendete Tasks wiederholt abgearbeitet werden, überprüft das CoMo-System nun die Sicherungen des Truth Maintenance Systems. Dabei wird in unserem Beispiel festgestellt, daß die Task ‘Entwickle Teil 1’ mitsamt ihrer Sub-Tasks und denselben Vorgabedaten bereits erfolgreich ausgeführt wurde. Diese Task braucht also nicht mehr neu gestartet zu werden, die zuvor gewonnenen Ergebnisse können weiterverwendet werden (Abbildung 3.51).

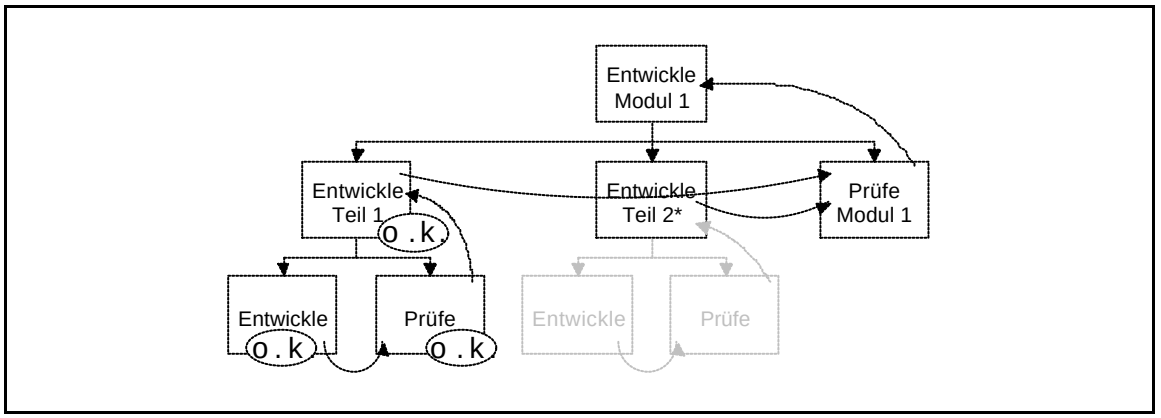


Abbildung 3.51: Beschreibung Backtracking - Bild 4

Neu gestartet werden muß also nur die Task ‘Entwickle Teil 2’ mit neuen Vorgabedaten. Sie kann daraufhin auch ihre Sub-Tasks wieder starten. Das vormals bereits berechnete Ergebnis der Sub-Task ‘Entwickle’ kann hier jedoch nicht wiederverwendet werden, da durch den geänderten Bauraum neue Vorgabedaten vorliegen. Die Task muß komplett neu bearbeitet werden.

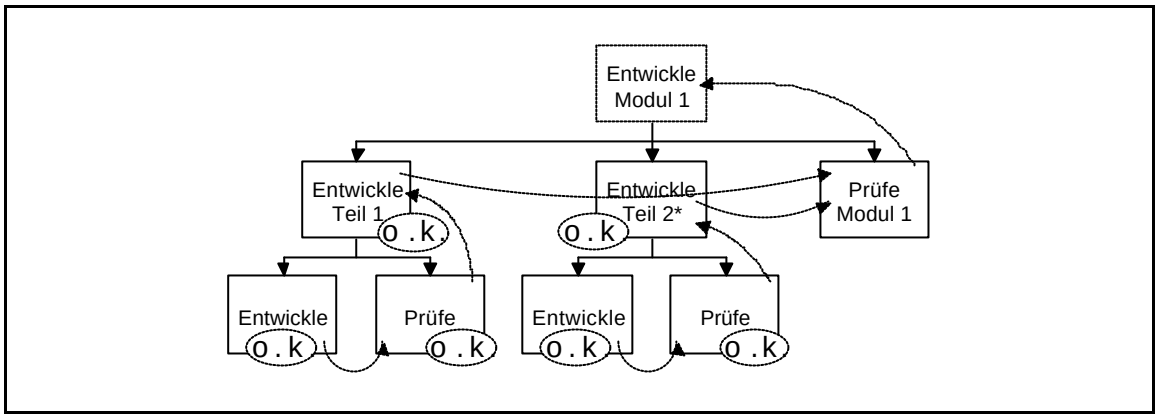


Abbildung 3.52: Beschreibung Backtracking - Bild 5

Wurde schließlich die Task ‘Entwickle Teil 2’, inklusive der Sub-Tasks, erfolgreich bearbeitet, kann durch die nun gültigen Datenfluß-Abhängigkeitsgleichungen die Task ‘Prüfe Modul 1’ abgearbeitet werden (Abbildung 3.52). Sie wird vollständig neu gestartet, da sie natürlich auch von den neuen Vorgabedaten abhängig ist. Erst nach einem erfolgreichen Abschluß dieser Task ist dann auch die übergeordnete Task ‘Entwickle Modul 1’ erfolgreich abgeschlossen (Abbildung 3.53).

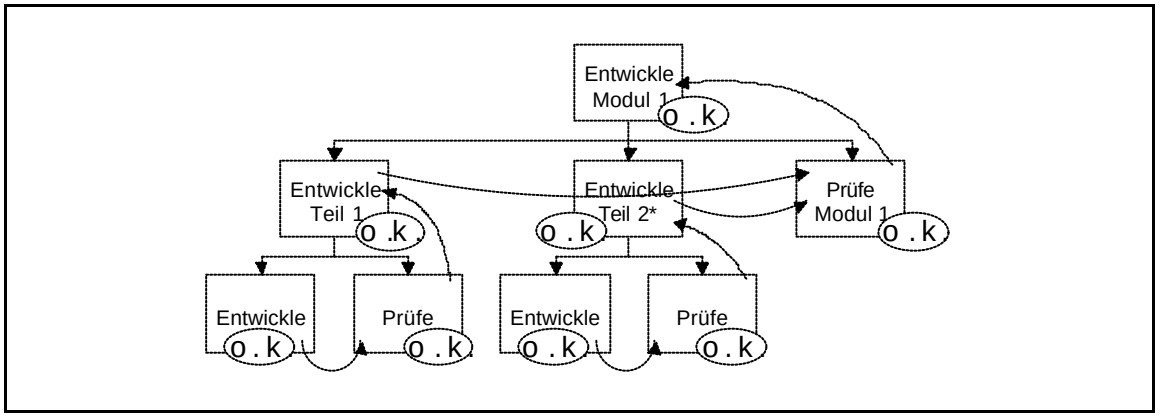


Abbildung 3.53: Beschreibung Backtracking - Bild 6

Dieses Beispiel hat nur einen kleinen Ausschnitt einer möglichen Task-Struktur betrachtet. Sollte der Fehler nicht erst in ‘Entwickle Modul 1’, sondern schon in einer wesentlich früheren Task verursacht worden sein, beispielsweise in der Task ‘Entwickle Teile’ des initialen Projektplans, ist natürlich ein wesentlich höherer Aufwand nötig. In diesem Fall wäre das Backtracking bis zur Task ‘Entwickle Teile’ im initialen Projektplan zurückgelaufen.

3.5.5 Abgrenzung und Bewertung

Bei einer genauen Betrachtung des CoMo-Kits stellt sich sicherlich die Frage, wie kann man dieses System einordnen. Ist es nun ein Projekt-Management-System oder eher ein Workflow-Management-System?

Gegen das Projekt-Management-System spricht das dafür typische Zeitmanagement, das bei CoMo-Kit keinerlei Beachtung findet. Auch Ressourcenmanagement wird nicht betrieben. Gegen das Workflow-Management-System spricht, daß CoMo-Kit durch die fehlende Kontrollflußmodellierung keine konkrete Unterstützung für stark strukturierte Prozesse bietet.
Da sich CoMo-Kit aber selbst als Unterstützung der Planung und Koordination komplexer, verteilter Entwicklungs- und Entwurfsprojekte versteht, wollen wir es eher den Projekt-Management-Systemen zuschlagen. Dazu passen auch die hauptsächlich vorgesehenen Anwendungsgebiete Software Engineering und Bebauungsplanung. Genau genommen, läßt es sich aber nicht exakt in die vorgefertigten Schubladen einordnen.

Für die von uns hauptsächlich betrachteten Produktentwicklungsprozesse ist CoMo-Kit, wie das vorige Beispiel zeigt, zwar prinzipiell geeignet, für eine perfekte Unterstützung fehlen ihm aber einige Eigenschaften. Die Hauptprobleme wären:

- kein Zeitmanagement,
- kein Simultaneous Engineering,
- keine Gruppenarbeitsmechanismen,
- keine explizite Kontrollflußmodellierung und
- bei der Anpassung an neue Vorgaben sind komplexe Aktionen nötig.

Durch das fehlende Zeitmanagement kann das System natürlich grundsätzlich keine Zeitangaben in den Projektplan übernehmen. Das System hat zwar einerseits keine Arbeit mit der Kontrolle von Zeitüberschreitungen und der Einhaltung eines Zeitplans. Andererseits sind aber gerade Zeitangaben für die Produktentwicklung unerläßlich, schließlich müssen Fertigungstermine eingehalten werden.

Simultaneous Engineering wird von CoMo-Kit ebenfalls nicht unterstützt. Zwar ist es denkbar, daß mehrere Tasks beziehungsweise Bearbeiter an ein und derselben Datenstruktur arbeiten. Eine konkrete Unterstützung dafür gibt es aber nicht. Weiterführende Mechanismen wie die Weitergabe vorläufiger Daten oder die direkte Anforderung von Daten existieren nicht.

Aus dem fehlenden Bearbeiten gemeinsamer Daten folgt dann natürlich, daß CoMo-Kit keine Unterstützung für Gruppenarbeitsmechanismen benötigt. Absprachen zwischen Tasks oder Bearbeitern sind also konventionell oder über andere Systeme zu tätigen.

Die Abhängigkeitsgleichungen des CoMo-Systems beschreiben ganz gut den Datenfluß zwischen Tasks, indem sie Eingabeparameter von den Ausgabeparametern anderer Tasks abhängig machen. Eine explizite Kontrollflußmodellierung gibt es nicht. Eine Task, respektive die darin enthaltene Methode, kann immer dann gestartet werden, wenn der ihr zugewiesene Bearbeiter dazu in der Lage ist. Beschränkungen gibt es höchstens durch die Angabe von Vor- und Nachbedingungen. So kann eine Vorbedingung den Start einer Task darauf beschränken, daß benötigte Daten in einer gewissen Qualität vorliegen müssen. Durch Nachbedingungen könnte eine Art Qualitätsstufen modelliert werden.

Ein weiterer Kritikpunkt für Produktentwicklungsprozesse ist sicher auch das Vorgehen von CoMo-Kit bei der Anpassung des Projektplans an neue Gegebenheiten. Prinzipiell wird dies durch die Konzeption der Abhängigkeitsgleichungen und das einfache Aufteilen in Teilprozesse gut unterstützt. Tritt indes in einer Task ein Fehler auf oder kann sie nicht gestartet werden, so schlägt sich dies durch den Backtracking-Mechanismus derart auf übergeordnete Tasks durch, daß Abhängigkeitsgleichungen ungültig werden und damit bereits erfolgreich abgeschlossene Tasks vollständig zurückgenommen werden müssen (siehe dazu das Beispiel in Kapitel 3.5.4). Etwas gedämpft wird dies durch den Einsatz eines Truth Maintenance Systems. Das TMS sichert die Ergebnisse erfolgreich abgeschlossener Tasks, so daß sie direkt wiederverwendet

werden können, wenn Änderungen am Projektplan sich nicht darauf auswirken. Dies verhindert effizient die wiederholte Ausführung bereits abgeschlossener Tasks. Noch nicht abgeschlossene Tasks müssen jedoch vollständig verworfen und neu gestartet werden. Dies ist für die zum Teil sehr langlebigen Prozesse einer Produktentwicklung natürlich nicht sinnvoll. Hier wäre es besser, Einzelergebnisse ungültig zu machen, anstatt ganze Tasks zurückzunehmen.

Der Backtracking-Mechanismus von CoMo-Kit ist gut geeignet für Anwendungen, in denen verschiedene Endergebnisse einer Aufgabe ausprobiert werden, wie zum Beispiel die bereits erwähnte Bebauungsplanung (siehe dazu [MaPe96]).

3.6 WoTel

WoTel steht für **Wo**rkflow und **Tel**ekooperation. Die Aufgabe des Projektes ist es, Anwendern von Workflow-Systemen zu zeigen, wie durch die Integration von Telekooperationsdiensten eine unternehmensweite kontinuierliche Vorgangsbearbeitung sowohl standortbezogen, als auch standortübergreifend realisiert werden kann (vgl. [WPS+97]).

Meist werden Workflow-Management-Systeme und Telekooperationsdienste isoliert betrachtet. Auf der einen Seite, den Workflow-Management-Systemen, wird eine Sequenz von arbeitsteiligen Aktivitäten unter den Teilnehmern koordiniert. In diesem Zusammenhang spricht man von asynchroner Zusammenarbeit, da man zeitversetzt beziehungsweise zumindest unabhängig voneinander arbeitet. Auf der Seite der Telekooperation werden gemeinsame zeitgleiche Sitzungen und Konferenzen unterstützt, weshalb man dies auch synchrone Zusammenarbeit nennt. Um jedoch schnelle und unmittelbare Reaktions-möglichkeiten ohne eine längerfristige Unterbrechung der Workflowbearbeitung zu erreichen, ist eine Integration unabdingbar. Dabei genügt es nicht, diese Dienste nebeneinander anzubieten, vielmehr ist eine enge Zusammenarbeit der Anwendungen notwendig.

Die Konzeption einer integrierten Anwendungsplattform ist das Ziel des WoTel-Projekts. Diese Anwendungsplattform soll von der Modellierung und Optimierung der Arbeitsabläufe über Workflow-Systeme bis zur räumlich verteilten Ausführung alle Ebenen miteinander verbinden. Der Schwerpunkt liegt jedoch in der Integration der Workflow-Ebene mit multimedialen Telekooperationsdiensten.

Durch diese Integration können während der Workflowbearbeitung Verzögerungen, durch nicht korrekt oder unvollständig bearbeitete Vorgänge, vermieden oder wenigstens reduziert werden. Diese Situationen erfordern, ohne moderne Telekooperationsdienste, meistens umständliche und zeitintensive Rückfragen und Diskussionen per Mail, Telefon oder sogar mittels persönlichen Treffen. Bessere Lösungen sind von einem Benutzer oder automatisch vom Workflow-System initiierte Telekonferenzen. Den beteiligten Benutzern wird hiermit die Möglichkeit gegeben, direkt mittels audiovisueller Kommunikation das Problem zu diskutieren. Dabei kann jede authorisierte Person, für alle sichtbar, Korrekturen und Veränderungen an einem Datenobjekt, beispielsweise einem Dokument, vornehmen.

3.6.1 Modellierung einer Sitzung

Der Vorschlag von WoTel zur Integration von Workflow und Telekooperation ist die Einbettung von Sitzungen in das Workflow-Modell. Dies bedeutet, daß synchrone Aktivitäten, die Kooperation von Benutzern betreffend, an entsprechenden Positionen in die formale Beschreibung der asynchronen Workflows aufgenommen werden müssen. Als Sitzung wird dabei beispielweise eine Konferenz bezeichnet, die über einen Telekooperationsdienst läuft.

Um nun ein Konferenz-System aus einem Workflow-System heraus einzusetzen, muß zunächst eine entsprechende Schnittstelle spezifiziert werden. Typische Schnittstellen-parameter sind:

- **Sitzungsname**: Eine Benennung der Sitzung, die auch zur Identifikation dient.

- **Teilnehmer**: Eine Liste von Adressen der Teilnehmer, die an der Sitzung teilnehmen. Die Liste sollte dynamisch sein, um es Teilnehmern zu ermöglichen, zu einer laufenden Sitzung hinzuzustoßen
- **Sitzungsleiter**: Meist auch der Initiator.
- **Tagesordnung**: Da hier eine Semantik dem System nicht bekannt ist, handelt es sich im Normalfall nur um einfache Zusatzinformationen.
- **Anlagen**: Während der Sitzung verwendete Dokumente und Programme. Dies kann vor Beginn der Sitzung bekannt sein und vom Workflow-Management-System vorbereitet werden.
- **Termin**: Konferenz-Systeme lassen gewöhnlich keine Verhandlungen und Festlegungen von Sitzungsterminen zu. Vom Workflow-Management-System sollte jedoch darauf geachtet werden, daß alle für die Sitzung notwendigen Teilnehmer zur Verfügung stehen.
- **Dienst**: Wenn Konferenz-Systeme mehrere Telekooperationsdienste zur Verfügung stellen, so kann hiermit angegeben werden, welcher Dienst benutzt wird.

Diese Parameter müssen vom Workflow-Management-System als Sitzungsbeschreibung geliefert werden, um eine Sitzung zu initiieren.

Ausgehend vom Workflow-System, erfolgt die Durchführung einer Sitzung in sechs Phasen (siehe Abbildung 3.54). Die Sitzung selbst (der gestrichelte Kasten) wird von zwei vorbereitenden Phasen eingeleitet und durch zwei abschließende Phasen beendet.

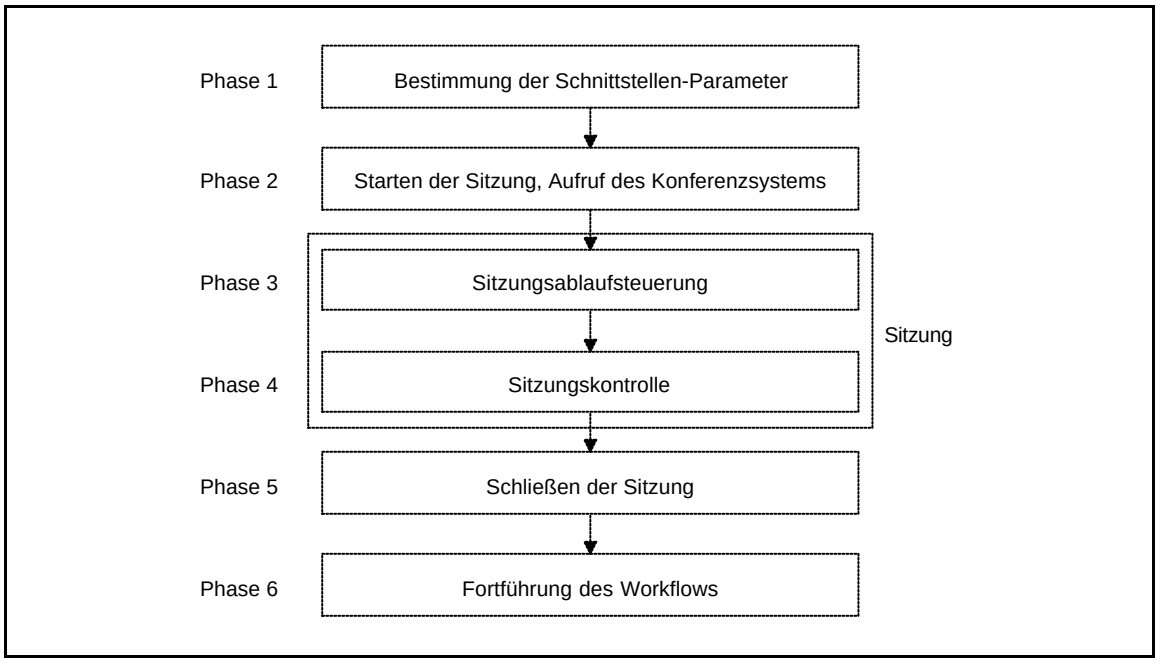


Abbildung 3.54: Integrationsphasen einer in ein WfMS eingebetteten Sitzung

In Phase 1 (Bestimmung der Schnittstellen-Parameter) wird die Sitzungsbeschreibung bereitgestellt. Dies kann sowohl manuell als auch systemseitig unterstützt erfolgen. Dabei können die Parameter direkt vor der Sitzung zusammengefaßt oder auch im Laufe vorhergehender Workflow-Aktivitäten gesammelt werden. Die Phasen 2 und 5 umschließen den generellen Sitzungsablauf. In Phase 2 muß dabei vor allem die gewonnene Sitzungsbeschreibung an das Konferenzsystem weitergeleitet werden. Die miteinander verwobenen Phasen 3 und 4

dienen der Unterstützung und Kontrolle während des Sitzungsverlaufs. In Phase 6 wird nach Beendigung der Sitzung der unterbrochene Workflow wieder fortgesetzt.

Vor dem Start einer Sitzung und dem damit verbundenen Durchlaufen dieser Phasen ist jedoch eine Modellierung im Workflow notwendig. Dabei muß zwischen geplanten und ad-hoc Sitzungen unterschieden werden.

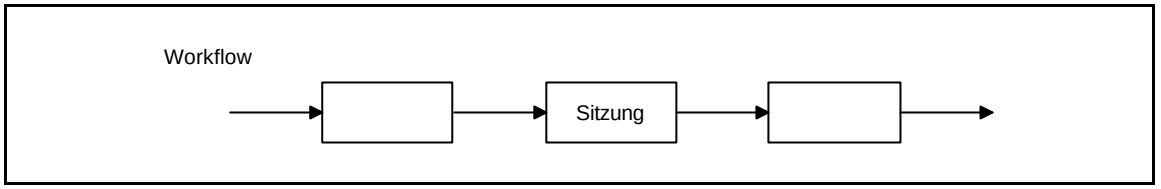


Abbildung 3.55: Geplante Sitzung

Geplante Sitzungen (Abbildung 3.55) können bereits während der Modellierung des Workflows an den gewünschten Stellen als Sitzungsaktivität plaziert werden. Erreicht der Workflow zur Ausführungszeit diese Aktivität, so werden die oben definierten Phasen durchlaufen. Dabei hängt es natürlich stark von der Sitzungsaktivität selbst ab, ob eine komplette Sitzungsbeschreibung bereits zur Modellierungszeit erfolgen kann, oder ob die Beschreibung vor dem Start der Sitzung noch ergänzt werden muß. Dies wäre beispielsweise dann der Fall, wenn Teilnehmer nur durch ihre Rollen modelliert werden, und zur Sitzung noch durch die tatsächlichen Personen ersetzt werden müssen. Eine Ergänzung könnte auch bei anderen Parametern der Sitzungsbeschreibung nötig werden, zum Beispiel bei den Anlagen. Diese könnten nämlich, obwohl in der Sitzung nötig, in vorhergehenden Aktivitäten des Workflows erst neu erstellt werden.

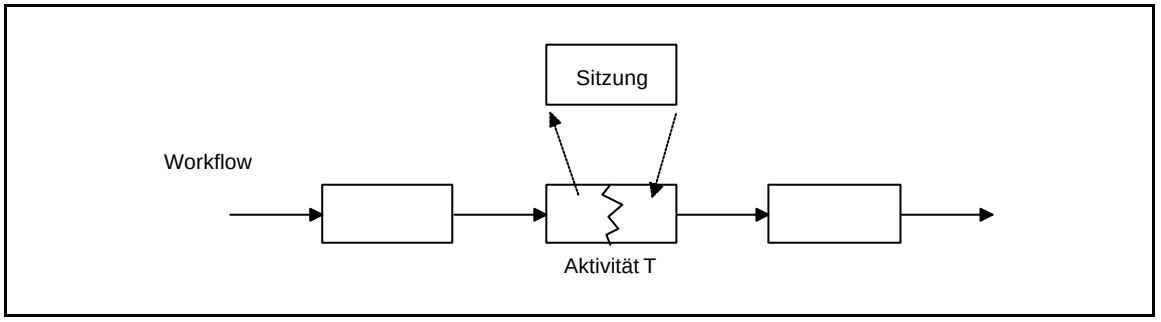


Abbildung 3.56: Ad-hoc Sitzung

Ad-hoc Sitzungen (Abbildung 3.56) erfolgen spontan an nicht vorgeplanten und zu rein zufälligen Zeitpunkten innerhalb eines Workflow. In der Modellierungsphase stehen noch keine Informationen darüber zur Verfügung. Um jederzeit aus einer Workflow-Aktivität heraus eine Sitzung zu initiieren, muß die Ressource Konferenz-System jederzeit bereitgehalten werden. Die Nutzung für spontane Sitzungen darf den Fluß des Workflow in keinem Fall behindern oder einschränken. Stellt sich jedoch heraus, daß eine Ad-hoc Sitzung immer wieder an der gleichen Stelle im Workflow auftritt, so kann sie bei einer neuen Modellierung in eine geplante Sitzung überführt und in den Workflow fest eingebaut werden.

Innerhalb einer Sitzung kann zwischen zwei weiteren Modellen unterschieden werden: statische Sitzungen und dynamische Sitzungen.

Statische Sitzungen haben einen vordefinierten Ablauf und können deshalb wie ein Workflow vor dem Start modelliert werden. Das Workflow-Management-System kann dadurch den Ablauf der Sitzung steuern, wobei die Aktivitäten nun aber nicht von einer Einzelperson, sondern von der

ganzen Gruppe der an der Sitzung Teilnehmenden durchgeführt werden. Statische Sitzungen können dabei sowohl geplant als auch spontan im Workflow stattfinden.

Bei **dynamischen Sitzungen** erfolgt keine Kontrolle durch das Workflow-Management-System. Sie entwickeln sich im Verlauf der Sitzung durch die Aktivitäten der Teilnehmer weiter und unterliegen auch der Verantwortung der Teilnehmer. Werden alle Aktivitäten einer dynamischen Sitzung in einer Art 'Black Box'-Aktivität zusammengefaßt, so könnte man dies als statische Sitzung mit nur einer Aktivität auffassen. In diesem Fall wären dynamische Sitzungen eine Untermenge der statischen Sitzungen.

3.6.2 Überwachung einer Sitzung

Eine Kontrolle und Überwachung des Sitzungsablaufs (Integrationsphase 4) ist notwendig, um die Ergebnisse der Sitzung wieder in den Workflow zu integrieren. Dafür wird eine Monitoringfunktion benötigt, die die Sitzungstätigkeiten protokolliert. Dieses Protokoll muß am Ende der Sitzung ausgewertet werden, um die Ergebnisse abzuleiten. Das Monitoring kann auf vier Ebenen erfolgen:

- kein Monitoring,
- manuelles Monitoring,
- halbautomatisches Monitoring und
- automatisches Monitoring.

Die Arten des Monitoring werden dabei hauptsächlich in zwei Punkten unterschieden: Wer zeichnet den Verlauf der Sitzung auf und wer ist für die Evaluierung, also für die Bestimmung der Nachfolgeaktivität anhand der Sitzungsergebnisse, verantwortlich. Am einfachsten läßt sich dies, wie in Tabelle 3.1 darstellen.

Art des Monitoring	Aufzeichnung der Sitzung	Evaluierung der Sitzung
kein Monitoring	keine Aufzeichnung	keine Evaluierung
manuelles Monitoring	manuell durch einen oder mehrere Teilnehmer, unterstützt durch eine Checkliste	manuell durch einen oder mehrere Teilnehmer, unterstützt durch eine Checkliste
halbautomatisches Monitoring	manuell durch einen oder mehrere Teilnehmer, anhand der vorgegebenen Checkliste	automatisch durch das Workflow-Management-System, anhand der vorgegebenen Checkliste
automatisches Monitoring	automatisch durch das Workflow-Management-System, anhand der vorgegebenen Checkliste	automatisch durch das Workflow-Management-System, anhand der vorgegebenen Checkliste

Tabelle 3.1: Aufzeichnung und Evaluierung einer Sitzung entsprechend der Art des Monitoring

Wird kein Monitoring durchgeführt, so wird das Konferenz-System vom Workflow-Management-System als 'Black Box' verstanden, über die keine internen Informationen zur Verfügung stehen. Dies ist eigentlich nur bei solchen Aktivitäten sinnvoll, wo der Workflow nicht vom Ergebnis der Sitzung abhängig ist, beispielsweise bei einer reinen Nachfrage bei einem Kollegen.

Halbautomatisches und automatisches Monitoring kann nur bei einer statischen Sitzung erfolgen, da dem Workflow-Management-System die Checkliste bekannt sein muß. Die kann aber nur dann vollständig sein, wenn vor Beginn der Sitzung alle Tätigkeiten der Sitzung bereits bekannt sind.

Eine **Checkliste** ist quasi als 'Schablone' für den Ablauf einer Sitzung zu verstehen. Während der Sitzung werden dort Tätigkeiten entsprechend ihres Bearbeitungszustands markiert. So läßt sich auch am Ende der Sitzung ein Gesamtergebnis ableiten.

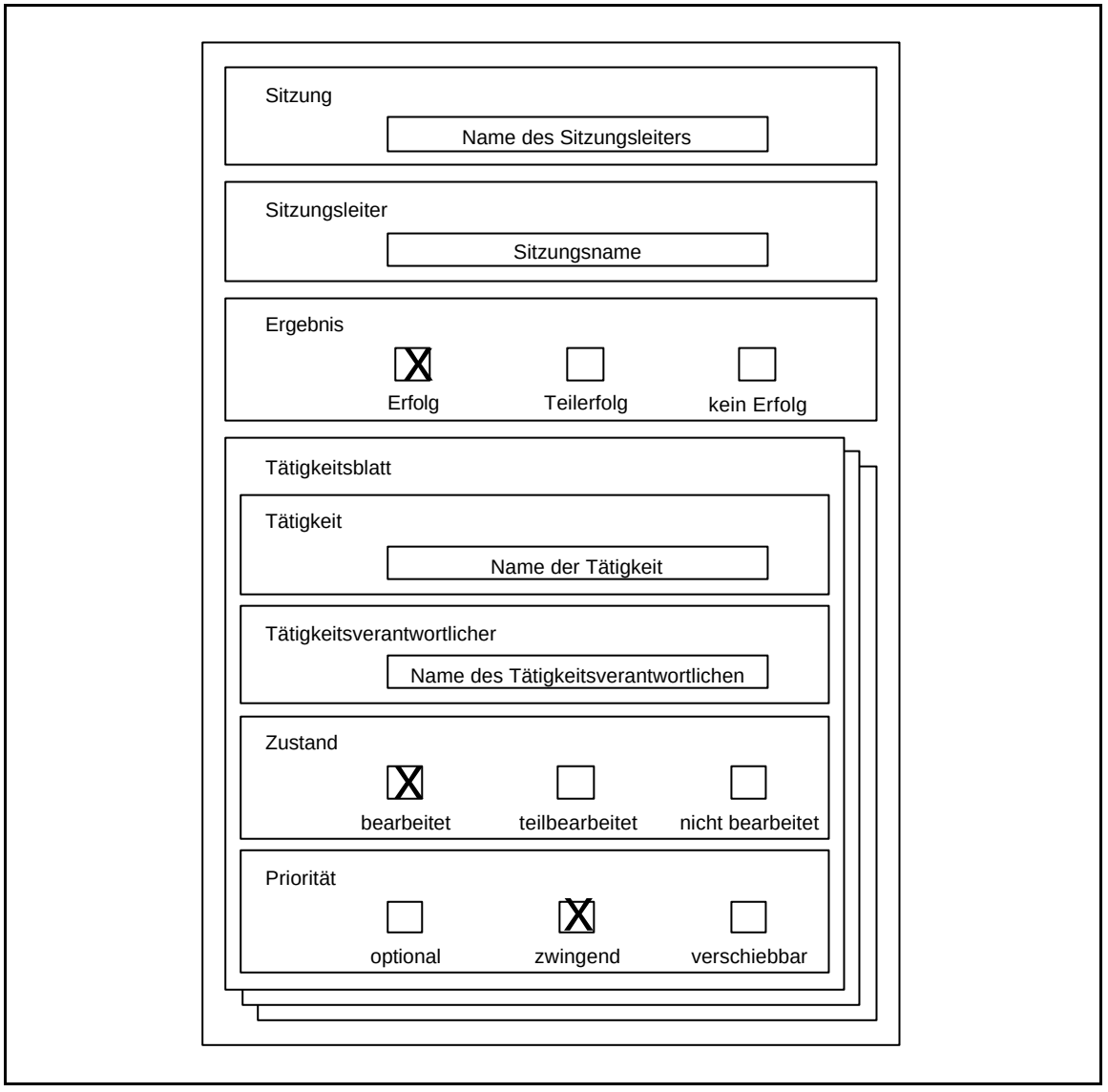


Abbildung 3.57: Checkliste für eine Sitzung

Checklisten werden in Abhängigkeit vom Sitzungsmodell, für das sie verwendet werden, erzeugt. Bei einer statischen Sitzung kann die Checkliste automatisch und direkt aus dem definierten Modell der Tätigkeiten der Sitzung erstellt werden. Dazu dient der sogenannte Checklistenagent. Er prüft alle ausgehenden Verbindungen einer Sitzungsaktivität auf weitere Sitzungsaktivitäten. Ist dies der Fall, überprüft er deren Teilnehmerlisten auf Überlappung. Solche Aktivitäten können gemeinsam in einer Sitzung erledigt werden. Disjunkte Teilnehmerlisten führen zur Erstellung von unabhängigen Checklisten, und damit zu unterschiedlichen Sitzungen. Bei dynamischen Sitzungen können während des Verlaufs neue Tätigkeitsblätter erzeugt und zur Checkliste hinzugefügt werden. Dafür wird ein Checklisteneditor benötigt, mittels dem eine spezifizierte

Person, meistens der Sitzungsleiter, in einem interaktiven Modus, die Checkliste erstellt oder ändert. Dieser Checklisteneditor kann natürlich auch für zusätzliche Arbeiten an Checklisten von statischen Sitzungen verwendet werden. Den Aufbau einer Checkliste mit mehreren Tätigkeitsblättern zeigt Abbildung 3.57.

Bei Beendigung einer Sitzung läßt sich anhand der Checkliste der Bearbeitungszustand der Tätigkeiten ermitteln. Dabei wird unterschieden zwischen **bearbeitet**, **teilbearbeitet** und **nicht bearbeitet**. Eine Tätigkeit ist teilbearbeitet, wenn sie zwar gestartet, aber nicht oder noch nicht vollständig beendet wurde. Sie ist nicht bearbeitet, wenn sie noch gar nicht gestartet wurde. Diese zwei letztgenannten Zustände können entstehen, wenn beispielsweise nicht alle zur Lösung der Aufgabe notwendigen Teilnehmer oder Anlagen verfügbar waren, oder die Lösung wegen fehlendem Konsens nicht gefunden wurde.

Desweiteren können einer Sitzungstätigkeit Prioritäten zugewiesen werden. Eine Tätigkeit ist **optional**, wenn sie bearbeitet werden kann, aber nicht muß. Sie ist **zwingend**, wenn sie vollständig bearbeitet werden muß. **Verschiebbare** Tätigkeiten müssen in der Sitzung nicht vollständig bearbeitet werden, sie können in späteren Sitzungen oder auch individuell fortgeführt werden.

Das Gesamtergebnis einer Sitzung ergibt sich letztendlich aus den Zuständen und Prioritäten der Tätigkeiten, so wie sie auf ihren entsprechenden Tätigkeitsblättern in der Checkliste vermerkt wurden. Dabei wird zwischen drei Kategorien unterschieden:

- **Erfolg**: Alle zwingenden und verschiebbaren Tätigkeiten sind bearbeitet. Der Workflow kann wie geplant weitergeführt werden.
- **Teilerfolg**: Alle zwingenden Tätigkeiten sind bearbeitet. Die verschiebbaren Tätigkeiten sind wenigstens teilbearbeitet. Hier ist eine zusätzliche Analyse notwendig, ob der Workflow mit dem Teilergebnis wie geplant fortschreiten kann, oder ob eine neue Sitzung einberufen werden muß.
- **kein Erfolg**: Mindestens eine zwingende Tätigkeit ist höchstens teilbearbeitet oder mindestens eine verschiebbare Tätigkeit ist nicht bearbeitet. Der Workflow kann so nicht weitergeführt werden. Die Sitzung sollte erneut abgehalten werden, eventuell mit zusätzlichen Teilnehmern oder Anlagen, um den Erfolg wahrscheinlicher zu machen.

3.6.3 Systemarchitektur

Eine Zielsetzung von WoTel ist es, möglichst ohne Änderungen an den verwendeten Workflow-Management-Systemen und Konferenz-Systemen auszukommen. Dazu wird ein sogenannter Konferenz-Broker als Mittler zwischen den beiden Systemen eingesetzt. Auf der einen Seite nimmt der Broker die Sitzungsbeschreibungen des Workflow-Management-Systems entgegen, wenn eine Sitzungsaktivität zur Ausführung ansteht, auf der anderen Seite übergibt er dem Konferenz-System die nötigen Aufrufparameter und startet damit die Sitzung. Er dient also quasi als definierte Schnittstelle zwischen den beiden Systemen und setzt die Ausgabe des Workflow-Systems in eine für das Konferenz-System passende Eingabe um (siehe Abbildung 3.58).

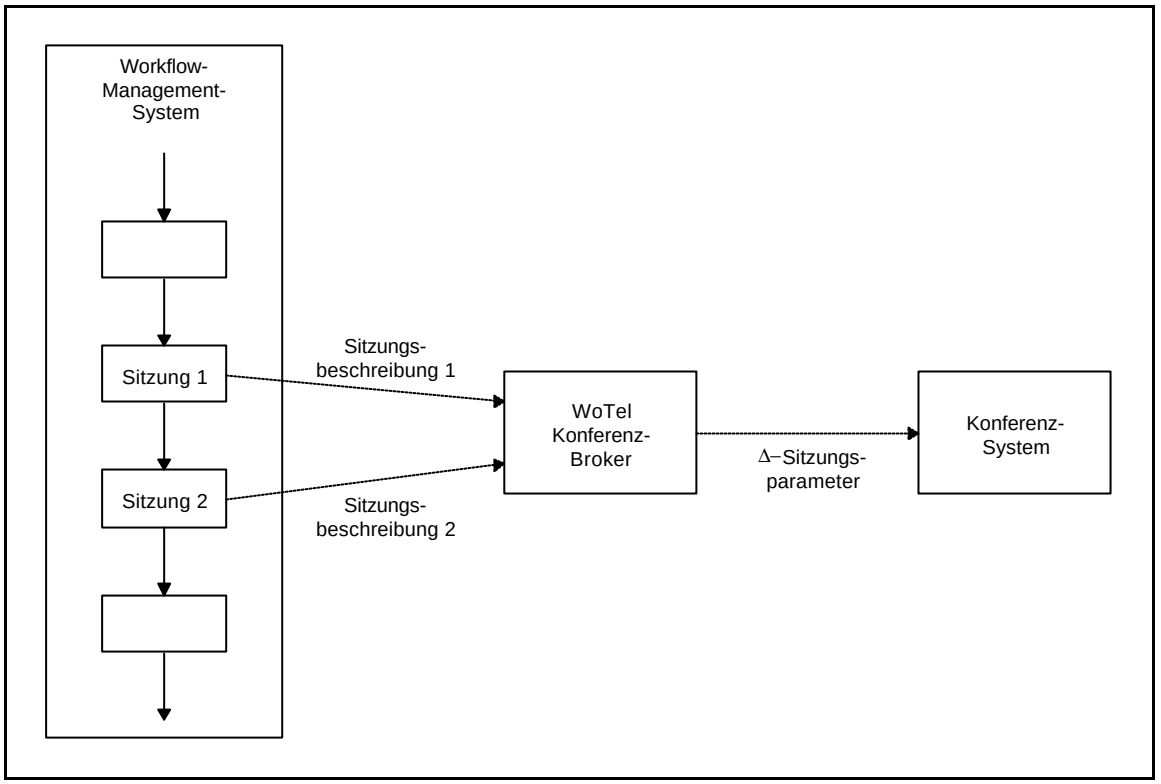


Abbildung 3.58: Konferenz-Broker vermittelt zwischen Workflow-Management-System und Konferenzsystem

Sitzungsbeschreibungen werden dabei inkrementell zusammengesetzt. Das heißt, bei direkt aufeinanderfolgenden Sitzungsaktivitäten extrahiert der Broker die Differenz der Beschreibungen als Δ-Information und überträgt nur diese zum Konferenz-System. Dadurch lassen sich die Teilnehmerschaft und die Anlagen dynamisch über mehrere Aktivitäten der Sitzung anpassen. Dieser Mechanismus wird als Δ-Konferenz bezeichnet. Ohne dieses Verfahren würden Sitzungen geschlossen, nur um sie kurz darauf mit eventuell nur leicht veränderten Parametern wieder neu zu starten.

Der Konferenz-Broker nimmt als virtueller Teilnehmer an der Sitzung teil. Dadurch ist er immer über den Zustand der Sitzung informiert und die anderen Teilnehmer sind sich bewußt, daß sie sich in einer in einen Workflow eingebetteten Sitzung befinden.

Workflow-Management-Systeme sind für gewöhnlich durch ihre Client/Server-Architektur, mit heterogenen Hardware- und Softwarekonfigurationen zwischen Server und Klienten, gekennzeichnet. Dagegen benötigen die meisten Konferenz-Systeme eine homogene Plattform unter den Teilnehmern. Bei ihnen finden sich, was die Verteilung angeht, sowohl Client/Server-Architekturen, als auch vollständig verteilte Architekturen.

Für den Aufbau einer WoTel-Architektur gibt es deshalb ebenfalls zwei Ansätze. Zum ersten Ansatz, der **Client/Server-Architektur** (Abbildung 3.59). Hier verfügt jeder Klientenrechner über einen WoTel-Klienten, der als Frontend zum Workflow-Management-System fungiert und den Checklisteneditor enthält. Außerdem leitet er die Sitzungsbeschreibungen zum WoTel-Server weiter. Dieser enthält den Broker, der mit dem Workflow-Management-Server und dem Konferenz-Server kommuniziert.

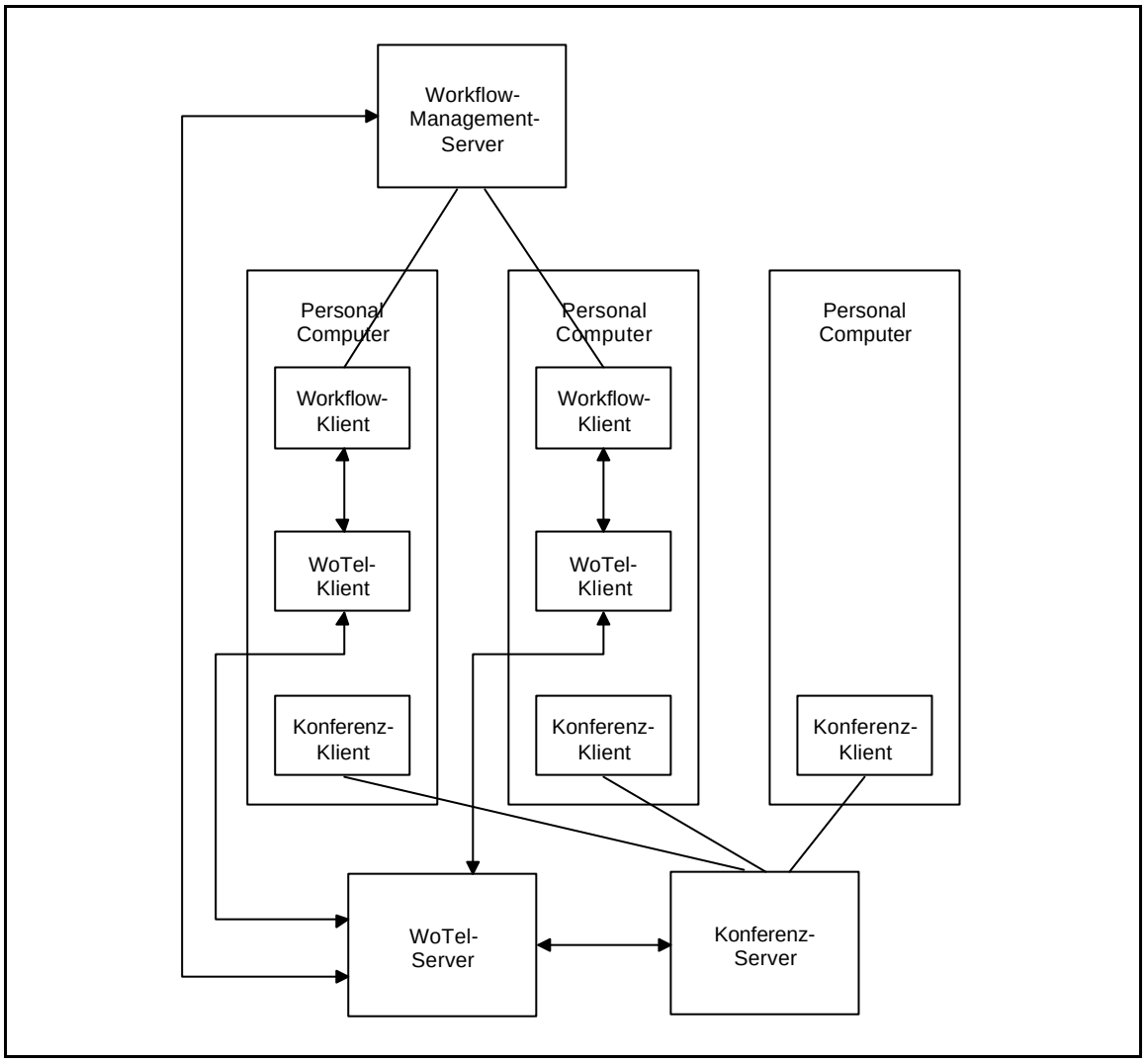


Abbildung 3.59: WoTel - Client/Server-Architektur

Die Vorteile dieser Architektur liegen in der konsistenten Haltung der Zustandsinformationen von Sitzungen, die nur einmal, nämlich im WoTel-Server vorliegen. Die Nachrichten-komplexität ist auf 1:n zwischen den Klienten und dem Server beschränkt. Ein Nachteil, der sich aus der Client/Server-Architektur ergibt, ist der Flaschenhals und der ‘Single-Point-of-Failure’, der durch den einzelnen Server entsteht, wenn das gesamte System höher skaliert wird.

Beim Ansatz für eine **vollständig verteilte Architektur** (Abbildung 3.60) wird die gesamte Funktionalität gleichermaßen auf die Arbeitsplatzrechner verteilt. Eine WoTel-Peer-Komponente übernimmt auf jedem Rechner die Aufgaben, für die im vorigen Modell der Klient und der Server zusammen verantwortlich waren.

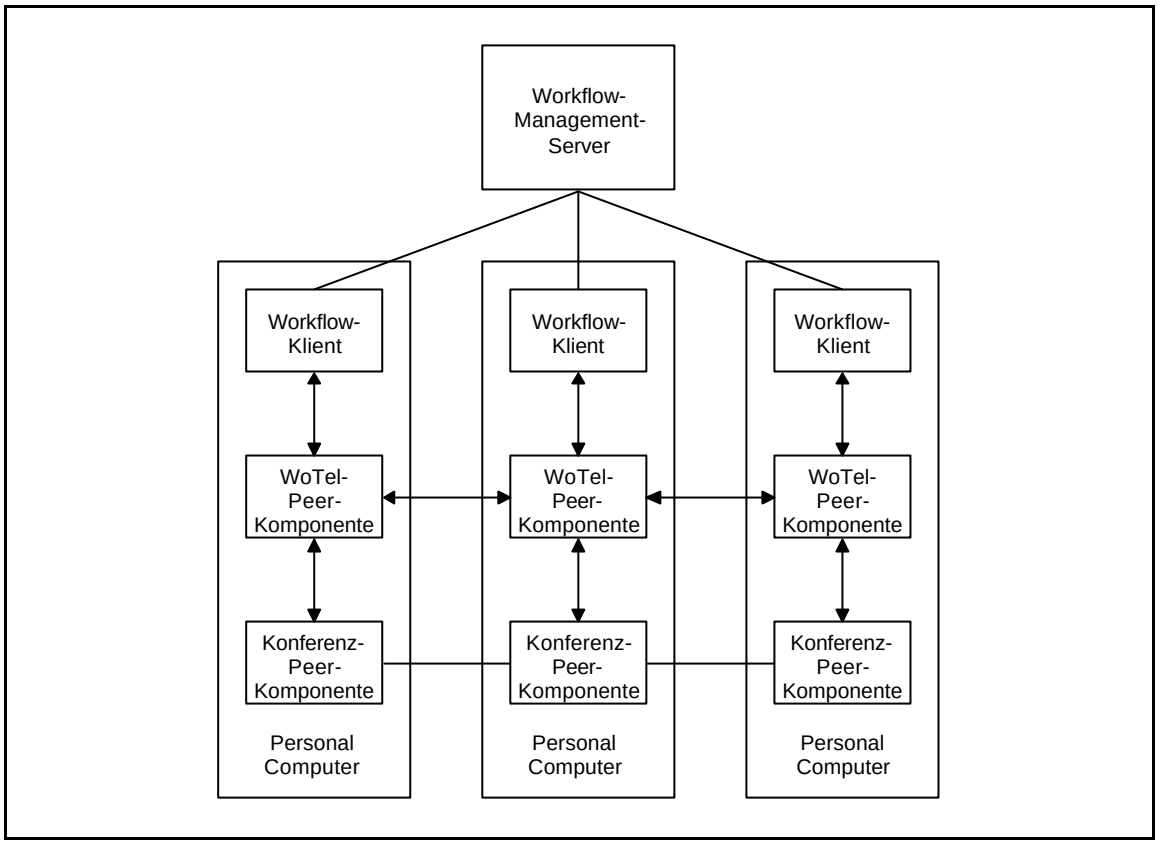


Abbildung 3.60: WoTel - vollständig verteilte Architektur

Durch diese Architektur werden zwar Kommunikationsengpässe vermieden, was zu einer höheren Skalierbarkeit des Systems führt, jedoch entsteht ein weit größeres Problem. Die Konsistenzhaltung der Sitzungsdaten zwischen den einzelnen Peer-Komponenten die an einer Sitzung beteiligt sind, wird um so komplizierter. Dadurch erhöht sich auch die Nachrichtenmenge die zwischen den Komponenten ausgetauscht werden muß.

3.6.4 Abgrenzung und Bewertung

Das Ziel des WoTel-Projekts ist nicht die Entwicklung eines neuen Workflow-Management-Systems. Daher kann es natürlich nicht direkt mit dem WEP-Workflow-Management-System verglichen werden. Auch die Beispieldarstellung eines Produktentwicklungsprozesses ist nicht möglich. WoTel kann als Groupware-Applikation verstanden werden. Es bietet eine Schnittstelle zwischen einem beliebigen Workflow-Management-System und einem beliebigen Telekooperations-Tool. Beide Programme bleiben dabei unverändert.

Ein Vergleich bietet sich mit der Kommunikations-Komponente von WEP an, der Konsolidierungsphase. Bei WEP ist diese Komponente ins System integriert, sie wird jedoch nicht wie bei WoTel als eigene Aktivität in den Workflowgraph eingebettet. Nach den, von WoTel für Sitzungen eingeführten Kategorien, ist eine WEP-Konsolidierungsrunde sowohl als Geplante Sitzung, als auch als Ad-hoc-Sitzung möglich. In WEP kann bei der Definition eines Meilensteins die Strategie bestimmt werden, wie Änderungen an vorzeitig weitergegebenen Daten behandelt werden. Wird hier der Modus CONSOLIDATION gewählt, so kann man die

bei der vorzeitigen Datenweitergabe entstehende Konsolidierungs-runde als geplant bezeichnen. In diesem Fall wird schließlich bereits zur Modellierungszeit festgelegt, daß eine Abstimmung über eine neue Objektversion zu erfolgen hat. Andererseits ist es nicht verboten, daß ein Bearbeiter auch ohne diesen Modus eine Konsolidierungsrunde einberuft. Dies kann dann zu einem völlig beliebigen Zeitpunkt erfolgen. In diesem Fall handelt es sich selbstverständlich um eine Ad-hoc-Sitzung.

Eine weitere Einteilung, die WoTel betreibt, ist die Unterscheidung zwischen statischer und dynamischer Sitzung. Eine WEP-Konsolidierungsrunde kann nicht direkt in eine dieser Klassifikationen eingeteilt werden. Generell hat sie sicher einen dynamischen Ablauf, da sie nicht vormodelliert wird und ihr interner Ablauf auch nicht vom System kontrolliert wird. Der interne Ablauf ist jedoch dahingehend festgelegt, daß das WEP-System die Operationen vorgibt und einen groben Ablauf für eine Konsolidierungsrunde vorschreibt. Wie sich diese WEP-Operationen den Integrationsphasen einer WoTel-Sitzung zuordnen lassen, zeigt Abbildung 3.61.

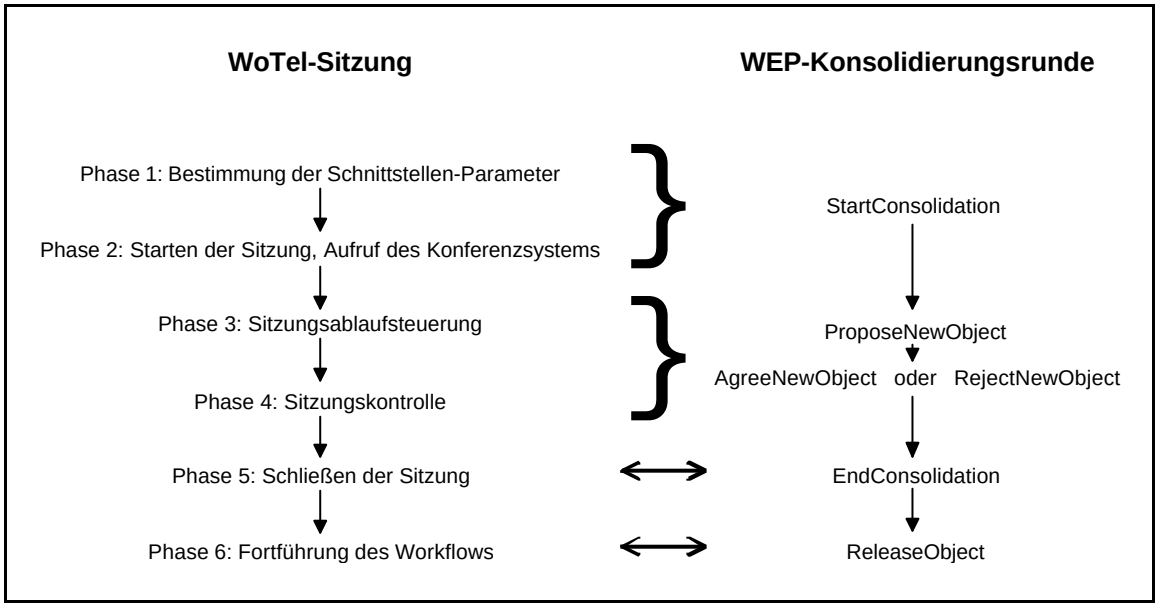


Abbildung 3.61: Vergleich im Ablauf einer WoTel-Sitzung und WEP-Konsolidierungsrunde

Die Kontrolle des Sitzungsablaufs (Monitoring) kann in WoTel in vier verschiedenen Variationen erfolgen. Zur Unterstützung wird dabei eine Checkliste eingesetzt, in der die Tätigkeiten der Sitzung entsprechend ihres Bearbeitungszustandes markiert werden. Dies erleichtert die Ableitung des Sitzungsergebnisses. WEP benötigt diesbezüglich eine andere Unterstützung, da von jedem Teilnehmer eine Entscheidung bezüglich Zustimmung oder Ablehnung zu einem vorgeschlagenen Objekt gefordert wird.

Im allgemeinen gehen die Darstellungsmöglichkeiten von WoTel, eine Sitzung betreffend, natürlich weit über den Funktionsumfang von WEP hinaus. Eine exakte Darstellung einer WEP-Konsolidierungsphase als WoTel-Sitzung ist aber dennoch nicht möglich, da WoTel hierfür nur einen Teilaspekt abdeckt. Dies wird durch die Definition eines Tätigkeitsblattes auf der Checkliste deutlich. Eine Sitzungstätigkeit in WoTel kann die Zustände ‘bearbeitet’, ‘teilbearbeitet’ und ‘nicht bearbeitet’ annehmen. Aus der Summe aller Zustände leitet sich das Ergebnis einer WoTel-Sitzung ab. In einer WEP-Konsolidierungsrunde kann ein teilnehmender Bearbeiter seine Meinung abgeben oder sich der Stimme enthalten. Der Zustand ‘teilbearbeitet’ existiert nicht. Stattdessen wird aber von jedem Teilnehmer die Information benötigt, wie er sich entschieden hat. Das Ergebnis einer Konsolidierungsrunde ist davon abhängig, wieviele

Teilnehmer zugestimmt beziehungsweise abgelehnt haben. Diese Zusatzinformation kann auf einer Checkliste nicht vermerkt werden.

WoTel bietet einen interessanten Ansatz für die Einbindung moderner Kommunikationsformen in bestehende Workflow-Management-Systeme. Der Hauptvorteil ist, daß bestehende Systeme ohne Änderungen weiterverwendet werden können. Dabei ist allerdings nicht ganz sicher, ob dies bei allen Kombinationen von Workflow-Management-Systemen und Konferenzsystemen so möglich ist. Aufgrund vieler unterschiedlicher Ansätze und Schnittstellen ist es sehr zweifelhaft, immer ohne Anpassungen auszukommen. Dies kann vielleicht zukünftig durch eine weitergehende Standardisierung, beispielsweise durch die Workflow Management Coalition (vgl. Kapitel 1.2.2), gewährleistet werden.

Auf der Basis der System-Architektur ist WoTel in jedem Fall gut gerüstet. Durch die Unterstützung der Client- / Server-Architektur und der vollständig verteilten Architektur, kann es sich jeweils an die Erfordernisse anpassen. Dabei ist dann nebensächlich, auf welchen Architekturen das Workflow-Management-System und das Konferenz-System basieren.

Durch den WoTel-Ansatz ist prinzipiell eine große Anzahl verschiedener Kommunikations-formen realisierbar. Manche spezialisierte Form, wie beispielweise die WEP-Konsolidierungsrunde, die zu eng ins System eingebunden ist, kann jedoch nicht vollständig realisiert werden.

3.7 Zusammenfassung

Nachdem nun die betrachteten Ansätze einzeln und detailliert beschrieben sind, möchte ich sie zum Abschluß in einer Gegenüberstellung zusammenfassen. Dabei werden besonders die Eigenschaften und Konzepte der Modelle in Bezug auf die Unterstützung der in Kapitel 1.4 entwickelten Anforderungen für Produktentwicklungsprozesse betrachtet. WoTel wird in die tabellarische Gegenüberstellung nicht mit einbezogen, da es sich dabei nicht um ein Modell handelt, das in der Lage ist, Arbeitsabläufe zu verwalten.

Tabelle 1 vergleicht die Mechanismen der Ansätze zur Unterstützung einer dynamischen Kontrollflußmodellierung. Dabei muß zwischen der Grobplanung und der Feinplanung eines Produktentwicklungsprozesses unterschieden werden. Die Grobplanung ist im Normalfall vormodelliert, sie gibt den groben Ablauf vor. Die Feinplanung ist von der zugrundeliegenden Objektstruktur der Produktdaten abhängig und muß sich zur Laufzeit darauf einstellen (vgl. Kapitel 1.3).

Tabelle 1: Unterstützung dynamischer Kontrollflußmodellierung

ADEPT	Der Kontrollfluß ist vormodelliert. Spezielle Operationen erlauben das Einfügen und Löschen von Aktivitäten zur Laufzeit. Dies erfolgt jedoch manuell.
CONCORD	Der Aufbau einer hierarchischen Aktivitätenstruktur erfolgt völlig dynamisch zur Laufzeit.
DYNAMITE	Die Weiterentwicklung eines Aufgabennetzes erfolgt dynamisch anhand eines Prozeßstruktur-Schemas, das die möglichen Kontrollflüsse definiert.
Procura	Ermöglicht einen dynamischen Aufbau einer Aufgabenhierarchie durch die Integration von Planung und Ausführung. Eine Kontrollflußmodellierung findet aber nicht statt. Die Anordnung der Aufgaben geschieht nur aufgrund der Datenflußabhängigkeiten.
CoMo-Kit	Ermöglicht einen dynamischen Aufbau einer Aufgabenhierarchie durch die Integration von Planung und Ausführung. Eine Kontrollflußmodellierung findet aber nicht statt. Die Anordnung der Aufgaben geschieht nur aufgrund von Abhängigkeitsgleichungen.
WEP	Der Kontrollfluß ist vormodelliert. Zur Laufzeit sind keine Änderungen möglich. Das Traversierungsmerkmal erlaubt aber automatisch das, von einer Objektbeziehung abhängige Aufsplitten in eine variable Anzahl paralleler Kontrollflüsse.

Tabelle 2 zeigt, ob die betrachteten Ansätze eine Unterstützung für unstrukturierte Teilprozesse zur Verfügung stellen. Dies ermöglicht dem Benutzer, statt dem Aufzwingen einer festen Arbeitsreihenfolge, eine individuelle und kreative Arbeitsgestaltung.

Tabelle 2: Unterstützung unstrukturierter Teilprozesse	
ADEPT	Eine Aktivität entspricht einem Programmschritt. Ein unstrukturierter Teilprozeß ist nur durch eine künstlich vorgegebene Reihenfolge von Aktivitäten, die zur Laufzeit manuell umstrukturiert wird, simulierbar.
CONCORD	Nicht möglich. Der Ablauf innerhalb einer Aktivität ist durch einen Workflow festgelegt.
DYNAMITE	Eine Aufgabe kann beliebig in Unteraufgaben zerlegt werden. Der prinzipielle Ablauf der Unteraufgaben wird aber wieder durch ein Prozeßstruktur-Schema bestimmt. Ein unstrukturierter Teilprozeß ist so nur durch ein Prozeßstruktur-Schema simulierbar, das alle möglichen Abläufe berücksichtigt.
Procura	Eine Aufgabe entspricht einem Programmschritt. Sie kann aber manuell in Unteraufgaben zerlegt werden, deren Reihenfolge nur von den Abhängigkeiten zwischen den Eingabeobjekten und Ausgabeobjekten abhängt.
CoMo-Kit	Nicht möglich. Eine Aufgabe entspricht einem Programmschritt. Sie kann in Unteraufgaben zerlegt werden, deren Reihenfolge aber durch Abhängigkeitsgleichungen festgelegt ist.
WEP	Zielorientierte Aktivitäten dienen als Sammlung mehrerer, in ihrer Reihenfolge nicht festgelegter Programmschritte. Jeder Benutzer kann dadurch seine Arbeitsweise selbst festlegen.

In Tabelle 3 werden Mechanismen gegenübergestellt, die eine zeitgleiche Zusammenarbeit (Simultaneous Engineering) der Benutzer eigentlich sequentieller Aktivitäten, an einem Datenobjekt oder an verschiedenen Versionen eines Objekts ermöglichen. Das simultane Bearbeiten sequentieller Prozeßschritte trägt zur Verkürzung von Entwicklungszeiten bei.

Tabelle 3: Unterstützung zeitgleicher Zusammenarbeit (Simultaneous Engineering)	
ADEPT	Nicht vorhanden. Eine Folgeaktivität wird immer erst dann gestartet, wenn die aktuelle Aktivität beendet ist.
CONCORD	Aktivitäten und Sub-Aktivitäten können gleichzeitig aktiv sein. Über einen Beziehungstyp kann eine Aktivität manuell vorzeitige Daten anfordern.
DYNAMITE	Beliebig viele Aufgaben können gleichzeitig aktiv sein und auch vorzeitige Daten liefern und bearbeiten.
Procura	Nicht möglich, da eine Unteraufgabe erst ausgeführt wird, wenn die in der Reihenfolge davorstehende bereits beendet ist. Außerdem werden übergeordnete Aufgaben erst dann fortgeführt, wenn alle Unteraufgaben beendet sind.
CoMo-Kit	Nicht möglich, da eine Unteraufgabe erst gestartet wird, wenn ihre Eingaben vorliegen. Übergeordnete Aufgaben werden während der Ausführung der Unteraufgaben gestoppt.
WEP	Durch die Erweiterung des Datenmodells um Qualitätsstufen und die Angabe von Meilensteinen (bestimmte Datenqualität zu bestimmtem Zeitpunkt) als Ziele einer Aktivität, wird vorzeitige Datenweitergabe möglich.

Tabelle 4 vergleicht die Ansätze im Hinblick auf die Unterstützung komplex strukturierter Objekte, wie sie für die Produktentwicklung typisch sind. Dabei muß während der Entwicklung auch der Zugriff auf Teile eines Objekts möglich sein.

Tabelle 4: Unterstützung komplex strukturierter Objekte

ADEPT	Nicht vorhanden. Globale Datenobjekte können im Datenfluß nur komplett Aktivitäten als Eingabeobjekt oder Ausgabeobjekt zugewiesen werden.
CONCORD	-[1]
DYNAMITE	-[1]
Procura	-[1]
CoMo-Kit	Objektorientiertes Datenmodell. Objekte können den Aufgaben aber nur komplett zugewiesen werden.
WEP	Objektorientiertes Datenmodell mit der Erweiterung um Qualitätsstufen. Ein Traversierungsmerkmal erlaubt die Aufsplittung globaler Datenobjekte an Relationen und so den Zugriff auf Teilobjekte.

In Tabelle 5 werden vorhandene Unterstützungen betrachtet, die zwischen den Benutzern eines Systems die Kommunikation und den Datenaustausch vereinfachen.

Tabelle 5: Unterstützung einfacher Kommunikationsmöglichkeiten zwischen Benutzern

ADEPT	-[1]
CONCORD	Spezielle Funktionen erlauben die Abstimmung über eine Zielspezifikation zwischen Sub-Aktivitäten.
DYNAMITE	-[1]
Procura	-[1]
CoMo-Kit	-[1]
WEP	Funktionen für eine Konsolidierungsphase erlauben den Vorschlag und die Abstimmung mehrerer Benutzer, die an verschiedenen Versionen von ein und demselben Objekt arbeiten, über eine Objektversion.

Vorhandene Unterstützungen für ein Zeit- und Ressourcenmanagement werden in Tabelle 6 dargestellt. Ein Zeitmanagement ist für das Modellieren und Einhalten der vorgegebenen Fristen eines Entwicklungsprozesses unbedingt erforderlich. Ein Ressourcenmanagement kann zugleich noch die Auslastung der Benutzer und Hilfsmittel in das Prozeßmanagement mit einbeziehen. Der Benutzer muß auf Zeitüberschreitungen aufmerksam gemacht und an Fristen erinnert werden.

Tabelle 6: Unterstützung von Zeit- und Ressourcenmanagement	
ADEPT	-[2]
CONCORD	-[1]
DYNAMITE	-[1]
Procura	Zeit- und Ressourcenmanagement überprüft bei der Zuweisung von Aufgaben an einen Benutzer dessen Verfügbarkeit und Auslastung. Ist ein ursprünglich zugewiesener Benutzer nicht mehr verfügbar, kann die Zeitplanung dynamisch an einen neuen Benutzer angepaßt werden.
CoMo-Kit	-[1]
WEP	Die Meilensteine einer zielorientierten Aktivität legen die Zeitpunkte fest, zu denen eine Entwicklung beendet sein muß. Das System beachtet diese Angaben und informiert den Benutzer bei Zeitüberschreitungen.

Außer dem WEP-Modell unterstützen alle anderen beschriebenen Ansätze nur teilweise die gestellten Anforderungen. Außer WEP ist jedoch auch kein anderer Ansatz speziell für Produktentwicklungsprozesse entworfen worden. Dennoch sind alle, wie die Beispiele in den vorigen Kapiteln gezeigt haben, prinzipiell in der Lage, Produktentwicklungsprozesse darzustellen.

[1] der vorliegenden Beschreibung nicht zu entnehmen oder nicht vorhanden

[2] noch Forschungsgebiet

4 Implementierung

Im Implementierungsteil der Arbeit war die Konzipierung und prototypische Implementierung der Serverkomponente des WEP-Workflow-Management-Systems gefordert. Die Serverkomponente sollte auf einem bestehenden System aufbauen und so ein bereits existierendes Datenhaltungssystem mitbenutzen. Der in Kapitel 2 beschriebene Funktionsumfang des WEP-Modells mußte dabei natürlich erhalten bleiben.

Neben der, durch das System bereitgestellten Basis-Funktionalität, war bei dessen Wahl natürlich auch die spätere Weiterentwicklung mit zu berücksichtigen. Dies beinhaltet zum Beispiel die flexible Realisierung eines Klienten oder einer Versionsverwaltung. Da das WEP-Workflow-Management-System im DaimlerChrysler Forschungszentrum in Ulm realisiert wird, fiel die Wahl auf Metaphase Version 3. Metaphase ist ein umfassendes Informations-Management-System und wird im Konzern in älteren Versionen bereits seit längerer Zeit benutzt. So konnte zu Beginn der Arbeit auf bestehendes Wissen zurückgegriffen werden. Metaphase bietet, selber aufbauend auf einem normalen Datenbank-system, ein objektorientiertes Daten- und Versionsmanagement. Die verteilte Client- / Server-Architektur unterstützt eine weite Palette von Plattformen und Datenbanken. Darüberhinaus bietet die neue Version 3.0 von Metaphase einen Zugriff über eine Web-orientierte Schnittstelle. In Kapitel 4.1 wird Metaphase, vor allem in Bezug auf die Implementierung einer Serverkomponente, näher beschrieben.

Im danach folgenden Kapitel 4.2 wird eine Architektur, aufbauend auf Metaphase, für das WEP-Workflow-Management-System und die Serverkomponente entwickelt. Kapitel 4.3 beschreibt die Funktionsschnittstelle der Serverkomponente, über die ein Klient auf die Operationen des WEP-Workflow-Management-Systems zugreifen kann.

4.1 Metaphase

Metaphase[1] ist ein System, das zum unternehmensweiten Informations-Management eingesetzt werden kann. Es soll die in einem Unternehmen entstandenen Informationen und Daten über alle Unternehmensvorgänge hinweg verwalten, vom Konzept bis zur Produktion. Metaphase bildet inzwischen die Grundlage für verschiedene auf dem Markt befindliche Softwareprodukte zu den Themen Produktdaten-Management (PDM), Elektronisches Dokumenten-Management (EDM), Konfigurations-Management (CM) und dergleichen.

Metaphase stellt an sich nur die Grundmechanismen für solche Systeme zur Verfügung. Die verteilte Client- / Server-Architektur unterstützt eine Vielzahl unterschiedlicher Plattformen und Datenbanken. Damit ermöglicht sie, von den verschiedensten Computersystemen eines Unternehmens aus, den Zugriff auf die gesamte gespeicherte Informations- und Datenbasis des Unternehmens. Das objektorientierte Datenmanagement von Metaphase bietet eine hohe Flexibilität in Bezug auf die Art der zu speichernden Datenobjekte. Für die Datenhaltung integriert Metaphase dafür ein ganz normales Datenbanksystem[2].

[1] Metaphase Enterprise wird entwickelt von der Structural Dynamics Research Corporation (SDRC).

[2] Unterstützt werden derzeit Oracle, Sybase, Informix und Microsoft SQL Server.

Der Zugriff auf die Funktionalität von Metaphase erfolgt im Normalfall über einen Klienten. Dies kann ein einfacher Kommandozeilen-Klient oder ein Fenster-Klient, basierend auf einer umfangreichen grafischen Bibliothek, sein. Seit Version 3 ist durch eine zusätzliche Komponente namens E!Vista der Zugriff mittels Web-Browser über einen Java-Klienten möglich. Dafür muß jedoch ein Web-Server mit voller Java-Unterstützung installiert sein[3]. Abbildung 4.1 zeigt den prinzipiellen Aufbau eines Metaphase-Systems.

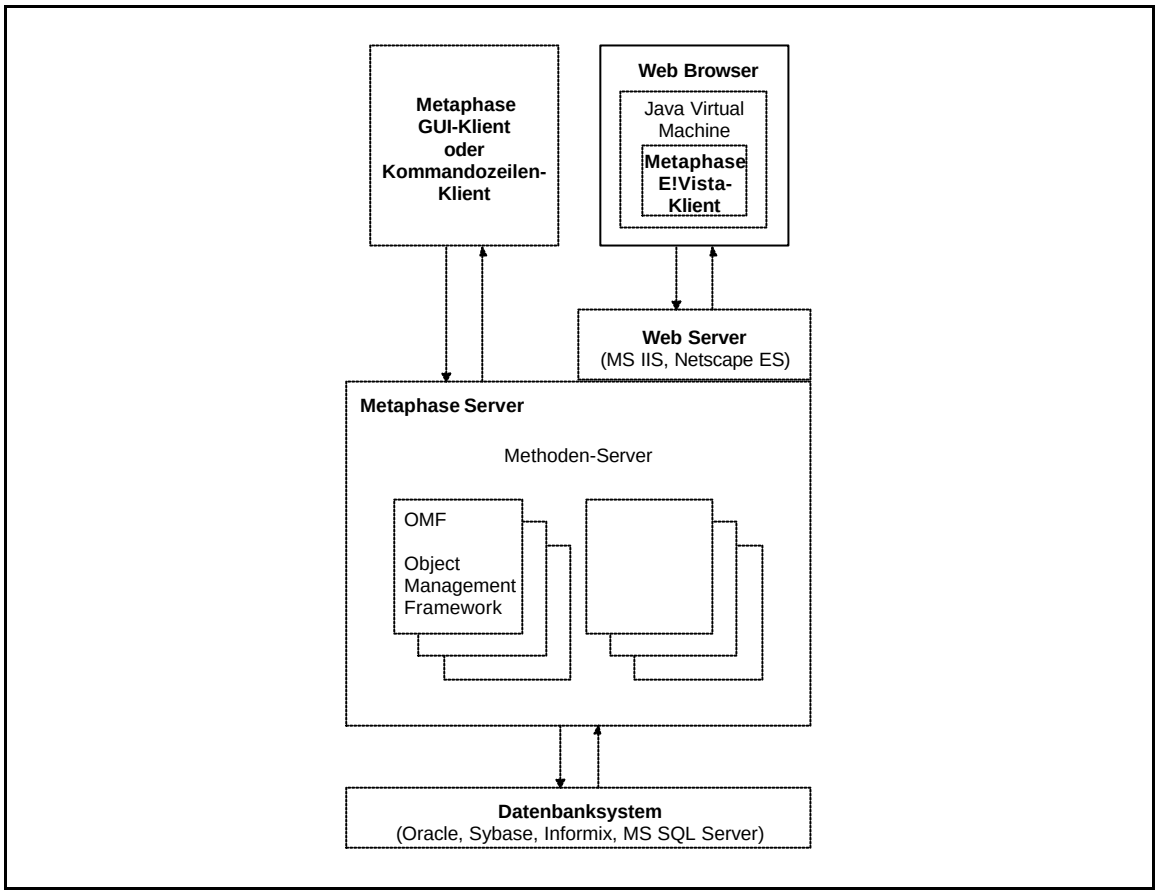


Abbildung 4.1: Prinzipielle Metaphase-Architektur

Die Hauptkomponente des Systems ist der Metaphase-Server. Er besteht aus einer Reihe verschiedener Module (Methoden-Server), die jeweils mit eigenen Funktionen (im Metaphase-Jargon Nachrichten genannt) ausgestattet sind und damit eine spezielle Funktionalität bieten. Das wichtigste Modul ist der OMF-Server (Object Management Framework). Er verwaltet und kontrolliert die Datenbasis und den Zugriff darauf (vgl. [Meta97a], [Meta97b]). Die verteilte Architektur erfordert natürlich noch einige weitere Komponenten, für die Kommunikation (mux) und der Zuordnung der eingehenden Nachrichten (dispatcher) zu dem entsprechenden Server. Diese Komponenten sind für unsere Betrachtung aber nicht von Belang und können als transparent angesehen werden.

Für die einfache Anpassung (Customizing) an fast beliebige Anwendungsbedürfnisse bietet Metaphase eine spezielle Komponente (Integrator Toolkit). Darüber sind verschiedene Methoden der Anpassung möglich (vgl. [Meta97c]). Zum einen können natürlich die Klienten entsprechend angepaßt oder programmiert werden. Zum anderen besteht die Möglichkeit, die von Metaphase zur Verfügung gestellten Funktionen und Objekte an die eigenen Bedürfnisse

[3] Unterstützt werden derzeit Netscape Enterprise Server und Microsoft Internet Information Server.

anzupassen und zu erweitern. Die beste Lösung ist aber sicherlich, einen weiteren Methoden-Server zu erstellen, der als Sammlung eigener Funktionen dient.

Ein selbst definierter Methoden-Server kann in jede bestehende Metaphase Installation eingebunden werden. Dazu wird er auf dem Rechnersystem, auf dem der Metaphase-Server betrieben wird, kompiliert und dann beim nächsten Start des Metaphase-Systems initialisiert.

4.2 Architektur

Die Architektur des WEP-Workflow-Management-Systems muß natürlich auf der zugrundeliegenden Metaphase-Architektur aufbauen. Die Serverkomponente wird in der Programmiersprache C als Methodenserver aufgebaut. So wird sie vollständig in den Metaphase-Server eingebunden und kann über die verteilte Architektur von jeder Metaphase-Client-Installation angesprochen werden. Abbildung 4.2 stellt diesen Sachverhalt dar.

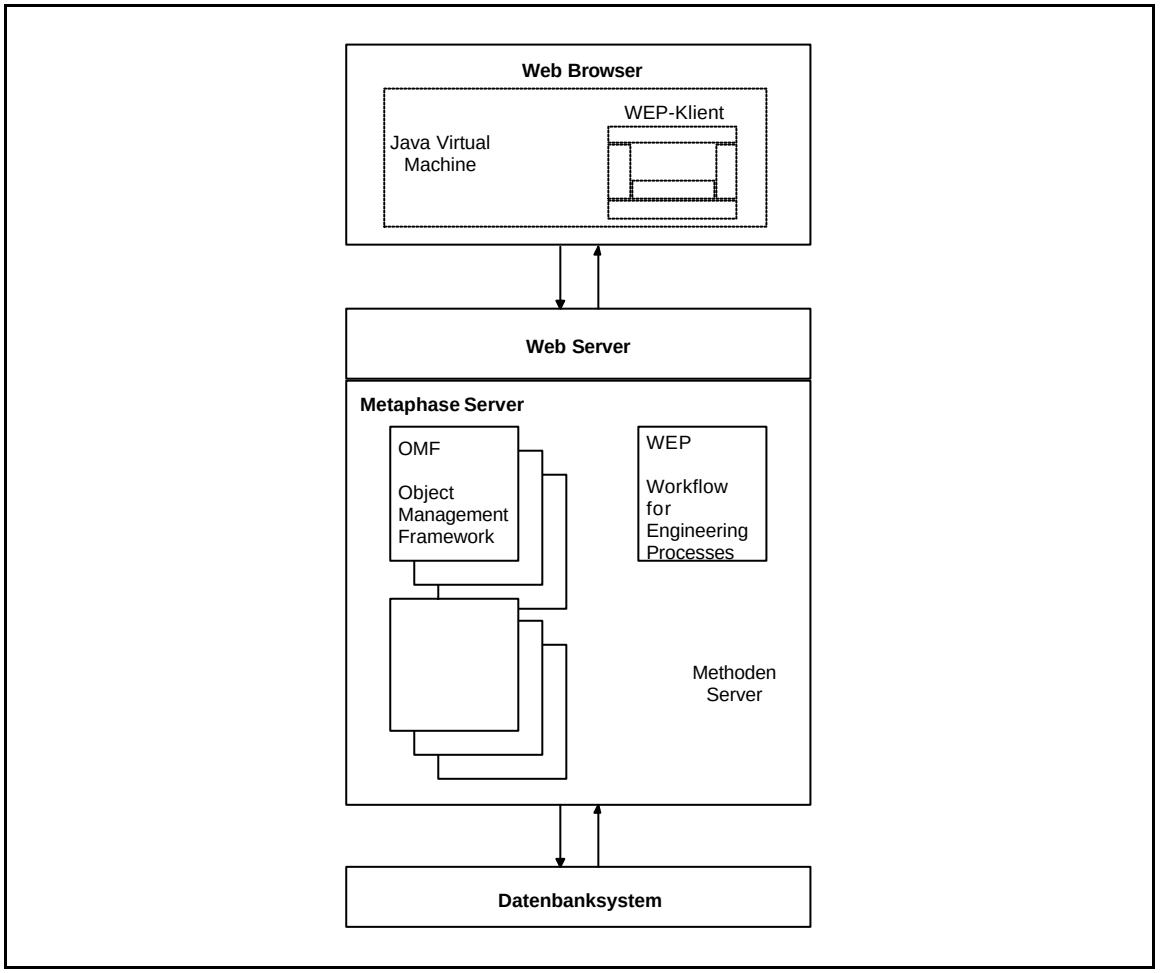


Abbildung 4.2: WEP-Server und WEP-Klient in der Metaphase-Architektur

Die Art des Klienten, der auf die Funktionalität des WEP-Servers zugreifen will, ist in diesem Fall belanglos. Am sinnvollsten ist es jedoch sicherlich, den WEP-Klient mittels der E!Vista-Komponente in Java zu realisieren. Dadurch bleibt er systemunabhängig und kann auf jeder Rechnerplattform über einen Web-Browser gestartet werden.

Für die Serverkomponente des WEP-Workflow-Management-Systems ergeben sich aus dieser Architektur hauptsächlich die folgenden zwei Vorteile:

- **Bestehende verteilte Umgebung**: Metaphase stellt ein verteilte Client- / Server-Architektur zur Verfügung, in die die Serverkomponente nahtlos eingefügt wird. Es muß kein Entwicklungsaufwand in die Unterstützung einer verteilten Anbindung investiert werden.
- **Integriertes Datenbanksystem**: Metaphase integriert ein bestehendes Datenbanksystem in seine Architektur und übernimmt die Verwaltung der darin gespeicherten Objekte. Darüberhinaus unterstützt es, unabhängig vom integrierten Datenbanksystem, ein objektorientiertes Datenmanagement. Der Zugriff auf die Objekte über Metaphase-Funktionen sowie die zusätzliche Versionsverwaltung vereinfachen den Datenbankzugriff.

Natürlich entstehen durch die Einbindung des WEP-Workflow-Management-Systems in Metaphase auch Nachteile, die man durch eine vollständige Eigenimplementierung umgehen könnte. Die Nachteile sind:

- **Unnötiger Ballast**: Metaphase ist ein umfassendes Paket zum Produktdaten-Management und bietet natürlich eine wesentlich höhere Funktionalität, als für das WEP-System benötigt wird. Dies bedingt natürlich einen deutlich größeren Ressourcen- und Performance-Bedarf, als eigentlich erforderlich.
- **Abhängigkeit**: Dadurch daß die Serverkomponente als Methodenserver in Metaphase eingebaut ist, ist man bezüglich des Datenaustauschs zwischen Server und Klient von Metaphase abhängig. So müssen komplexere Daten vor der Übergabe erst in einen für Metaphase bekannten Objekttyp umgewandelt beziehungsweise eingetragen werden. Dasselbe Problem betrifft den Datenaustausch zwischen dem Server und dem Datenbanksystem. Auch hier muß die Vorgehensweise an die Möglichkeiten von Metaphase angepaßt werden.

Diese Nachteile müssen natürlich immer im Vergleich mit dem erhöhten Mehraufwand einer vollständigen Eigenimplementierung gesehen werden. Außerdem stellt diese Implementierung der WEP-Serverkomponente nur einen Prototyp dar, der die Anwendbarkeit der Konzepte des WEP-Modells in der Praxis darstellen soll.

Um die Nachteile dennoch etwas einzuschränken, wurde versucht, die Serverkomponente so unabhängig wie möglich zu konzipieren. Abbildung 4.3 zeigt die zwei Berührungspunkte zwischen der WEP-Serverkomponente und Metaphase. Nur der Datenaustausch mit Klienten und der Zugriff auf Datenobjekte läuft über Metaphase. Alle anderen Komponenten innerhalb des Servers kommen ohne Metaphase-Funktionen aus.

In Abbildung 4.3 ist darüberhinaus das Gesamtkonzept des WEP-Workflow-Management-Systems dargestellt. Es besteht aus einer Modellierungs-Komponente und einer zweiteiligen Laufzeit-Komponente. Mit der Modellierungs-Komponente kann mit grafischer Unterstützung die Workflow-Beschreibung generiert werden. Die Laufzeit-Komponente untergliedert sich in den Server und einen oder mehrere Klienten. Für uns ist hauptsächlich der Server von Interesse. Die Modellierungskomponente und der Klient sind Teil weiterführender Arbeiten.

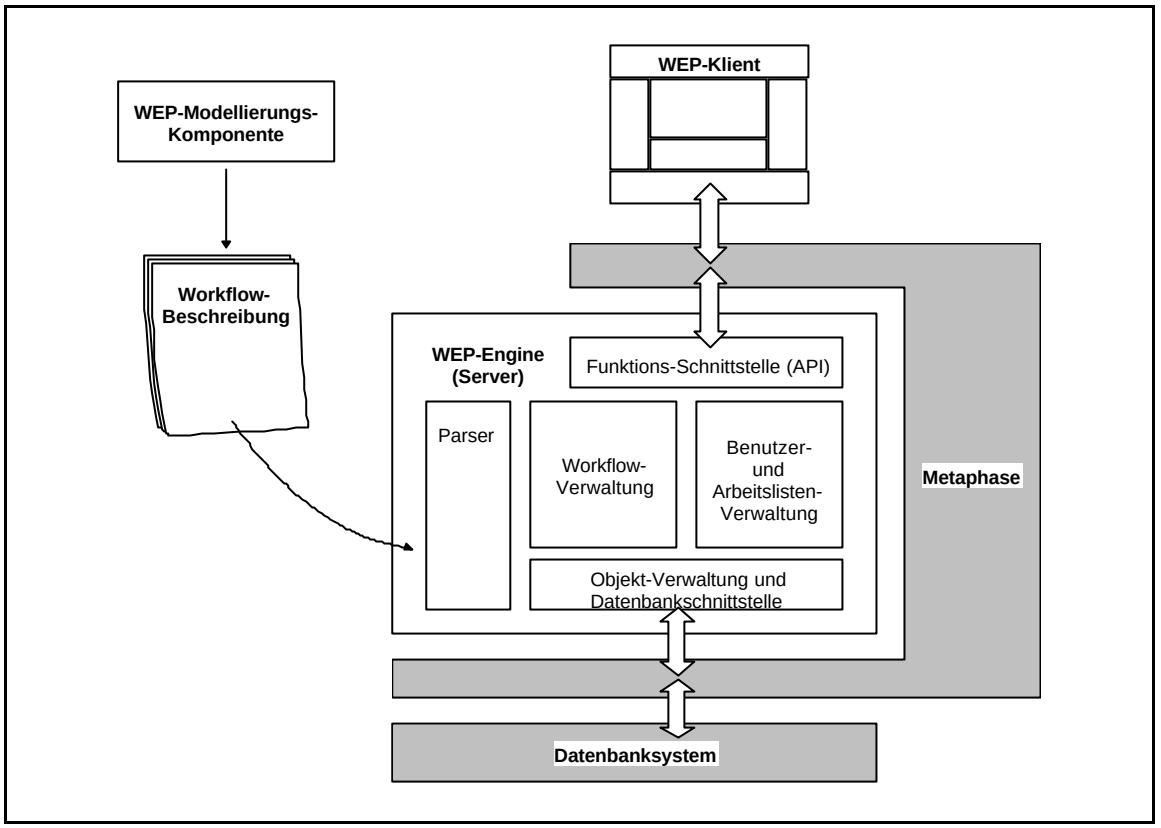


Abbildung 4.3: WEP-Architektur

Der Server besteht aus mehreren Modulen, die im folgenden näher beschrieben werden:

- **Parser**: Das Parsermodul nimmt die Datei mit der Workflow-Beschreibung entgegen. Es parst die Beschreibung und legt die, für die Ausführung des Workflows notwendigen Daten, in einer Struktur ab, aus der sie später ohne großen Aufwand direkt ausgelesen werden können. Die anderen Module, mit Ausnahme der Funktions-Schnittstelle greifen auf diese Daten zu. Die Syntax der Workflow-Beschreibung ist in Kapitel 2 genau erklärt. Im Anhang wird sie nochmals vollständig in Form von Syntaxdiagrammen dargestellt.
- **Funktions-Schnittstelle (API)**: Die Funktions-Schnittstelle stellt der Außenwelt die Funktionalität der Servers zur Verfügung. Diese Funktionen können von einem Klienten aus direkt aufgerufen werden. Die Funktionen wurden in Form von Metaphase-Nachrichten implementiert. Eine detaillierte Beschreibung findet sich in Kapitel 4.3.
- **Benutzer- und Arbeitslisten-Verwaltung**: Dieses Modul übernimmt alle Aufgaben bezüglich der Benutzer des WEP-Workflow-Management-Systems und ihrer Arbeitslisten. Meldet sich ein Benutzer an, so wird er in der Benutzerliste eingetragen. Meldet er sich ab, wird er wieder ausgetragen. Für jeden angemeldeten Benutzer wird eine Arbeitsliste (Worklist) erstellt. In ihr werden die Aktivitäten eingetragen, die vom Benutzer aufgrund seiner Rollenzuordnung bearbeitet werden können. Außerdem enthält sie Informationen bezüglich vorzeitig weitergegebener Objekte, zurückgeforderter Objekte, Konsolidierungsphasen und dergleichen. Dem Benutzer steht eine Funktion zur Verfügung, mit der er den Inhalt seiner Arbeitsliste ständig auslesen kann (Polling).
- **Workflow-Verwaltung**: Die Workflow-Verwaltung ist das zentrale Modul der Serverkomponente. Es erstellt den Abhängigkeitsgraphen eines Workflows und verwaltet alle Daten bezüglich der darin enthaltenen Aktivitäten. Die Workflow-Verwaltung kontrolliert und überprüft die Ausführung einer Aktivität, die Ausführung eines Programmschrittes

(Werkzeugs), die Anforderung und Freigabe von Objekten, sowie die Mechanismen der vorzeitigen Datenweitergabe. Darüberhinaus muß sie das Einhalten der, in der Workflow-Beschreibung definierten Zeitangaben und Fristen überwachen und die geforderten Qualitätsstufen von Meilensteinen mit den tatsächlichen Qualitätsstufen von erstellten Objektversionen vergleichen. Treten im Ablauf des Workflows Änderungen ein, die einen Benutzer betreffen, so werden diese Änderungen in dessen Arbeitsliste eingetragen. Dies geschieht in Zusammenarbeit mit dem Modul der Benutzer- und Arbeitslisten-Verwaltung. Die Workflow-Verwaltung wird hauptsächlich vom Klienten über Funktionsaufrufe gesteuert.

- **Objekt-Verwaltung und Datenbankschnittstelle**: Dieses Modul verwaltet über Referenzen die Objekte, die in der Datenbank abgelegt sind. Der direkte Zugriff auf die gespeicherten Objekte erfolgt über Metaphase-Funktionen. Alle innerhalb des Workflows benutzten Objekte werden in einer Liste eingetragen. Zur Kontrolle der Abhängigkeiten zwischen verschiedenen Aktivitäten, die mehrere Versionen von ein und demselben Objekt verwenden können, muß genau über die Versionserstellung Buch geführt werden.

4.3 Beschreibung der Funktionsschnittstelle

Die Funktionsschnittstelle orientiert sich stark an den Operationen der in Kapitel 2.8 beschriebenen Benutzerinteraktionen. Eine Implementierung benötigt natürlich noch einige zusätzliche Funktionen. Für die Prototyp-Implementierung wurde jedoch versucht, mit insgesamt möglichst wenig Funktionen auszukommen.

Auf der Workflow-Ebene sind zwei Funktionen verfügbar. **InitWorkflow** (Kapitel 4.3.1) initialisiert und startet einen neuen Workflow, mit **CancelWorkflow** (Kapitel 4.3.2) wird ein Workflow wieder beendet, beziehungsweise abgebrochen.

Für die Benutzer-Ebene gibt es zwei entsprechende Funktionen. So kann man mit **RegisterUser** (Kapitel 4.3.3) dem System einen neuen Benutzer bekannt machen. Die gegensätzliche Funktion **CancelUser** (Kapitel 4.3.4) meldet einen Benutzer wieder vom System ab. Zusätzlich gibt es noch die Funktion **PollWorkList** (Kapitel 4.3.5). Mit ihr kann ein Benutzer den Inhalt seiner Arbeitsliste (Worklist) abfragen.

Der Zugriff auf eine Aktivität wird über fünf Funktionen gesteuert. **StartActivity** (Kapitel 4.3.6) erlaubt einem Benutzer, eine in seiner Arbeitsliste eingetragene Aktivität zu starten. Während der Arbeit an der Aktivität kann die Ausführung für eine gewisse Zeit mit der Funktion **SuspendActivity** (Kapitel 4.3.7) unterbrochen und mit **ResumeActivity** (Kapitel 4.3.8) wieder fortgesetzt werden. Hält ein Bearbeiter seine Arbeit an der Aktivität für beendet, so ermittelt er mit der Funktion **PrepareFinishActivity** (Kapitel 4.3.9) die erlaubten Rückgabewerte der Aktivität. Aus diesen Werten sucht er sich einen, seiner Meinung entsprechenden aus, und meldet ihn per **FinishActivity** (Kapitel 4.3.10) dem System. Mit diesem Rückgabewert als Resultat wird die Aktivität dann wirklich beendet.

Zur Bearbeitung einer Aktivität benötigt der Bearbeiter Zugriff auf Datenobjekte. Dies erreicht er über die Funktion **FetchObject** (Kapitel 4.3.11). Sie erzeugt eine neue und für ihn änderbare Objektversion. Nach einer Bearbeitung des Datenobjekts kann der Bearbeiter mit **ReleaseObject** (Kapitel 4.3.12) eine vorzeitige oder fertige Version freigeben. Mit **RevokeObject** (Kapitel 4.3.13) kann er eine so freigegebene Objektversion wieder zurücknehmen. Benötigt der Bearbeiter selber unbedingt eine neue Version eines Eingabe-objekts, so kann er dieses mit der Funktion **RequestObject** (Kapitel 4.3.14) anfordern.

Die Bearbeitung von Datenobjekten geschieht mit vordefinierten Programmschritten über Werkzeuge (Anwendungen). Diese werden innerhalb einer Aktivität über die Funktion **StartProgramStep** (Kapitel 4.3.15) gestartet und mit **FinishProgramStep** (Kapitel 4.3.17) wieder beendet. Damit kann auch das Ergebnis der Programmausführung ermittelt werden. Mit der Funktion **PollProgramStep** (Kapitel 4.3.16) kann bei langdauernder Ausführung festgestellt werden, ob der Programmschritt bereits beendet ist oder noch läuft.

Für die Unterstützung der Gruppenarbeit, beziehungsweise Konsolidierungsrunde, stehen fünf Funktionen bereit. **StartConsolidation** (Kapitel 4.3.18) beruft eine Konsolidierungsphase ein. **EndConsolidation** (Kapitel 4.3.22) beendet sie wieder. Während der Konsolidierungsphase kann durch den Benutzer, der sie einberufen hat, eine neue Objektversion vorgeschlagen werden. Dies geschieht über die Funktion **ProposeNewObject** (Kapitel 4.3.19). Alle anderen beteiligten Benutzer müssen dieser Objektversion mit **AgreeNewObject** (Kapitel 4.3.20) zustimmen oder sie mit **RejectNewObject** (Kapitel 4.3.21) ablehnen.

Die in den Unterkapiteln folgende detaillierte Beschreibung aller Funktionen orientiert sich an der von Metaphase geforderten Syntax. Die Funktionen wurden als externe Metaphase-Nachrichten implementiert. Nur externe Nachrichten können von Klienten aufgerufen werden. Dazu war es erforderlich eine Klasse für den Server zu definieren und dieser Klasse, die als extern deklarierten Nachrichten, hinzuzufügen:

```
define class WepRoot parent is PdmRoot;

attach class message w0InitWorkflow to WepRoot in server wepsvr;
...

define external class message w0InitWorkflow
(
   input  : string    classname    ::
   input  : string    workflowfile ::
   input  : string    endtime      ::
   output : string *  workflowname ::
   output : integer * errorcode    ::
   output : integer * mfail
)
...
```

Bei der Deklaration der Nachrichten müssen natürlich die Bedingungen von Metaphase eingehalten werden. Um doppelte Nachrichtennamen zu vermeiden, sollte ein freier Präfix vor den Funktionsnamen gestellt werden. Für WEP wurde *w0* gewählt. Der Kopf einer Metaphase-Nachricht definiert die Parameter der Funktion. Die Parameter werden über einen RPC-ähnlichen, von Metaphase gesteuerten Mechanismus, übergeben. Dadurch sind nur vordefinierte Variablentypen zulässig und es muß zusätzlich definiert werden, ob es sich um einen Eingabe- oder Ausgabeparameter handelt.

Die Parameter *classname* und *mfail* sind für alle Klassen-Nachrichten vorgeschrieben. *classname* sollte der Name der Klasse zugewiesen werden, die für die Bearbeitung verantwortlich ist. *mfail* bekommt als Rückgabe einen Fehlerstatuscode von Metaphase. Allen WEP-Nachrichten wurde zusätzlich der Ausgabeparameter *errorcode* hinzugefügt. Er liefert eine vom WEP-Server definierte Fehlernummer zurück.

4.3.1 InitWorkflow

Initialisiert und startet einen neuen Workflow.

Definition der Metaphase-Methode:

```
message WepRoot:w0InitWorkflow
(
   input  : string    classname    ::
   input  : string    workflowfile ::
   input  : string    endtime      ::
   output : string *  workflowname ::
   output : integer * errorcode    ::
   output : integer * mfail
)
```

Parameter:

`input : string workflowfile`
: Pfad und Dateiname der Workflow-Beschreibung. Die Datei sollte vom Server aus erreichbar sein.

`input : string endtime`
: Enddatum und -zeit, zu der der Workflow fertig bearbeitet sein muß. Im Format „DD.MM.YYYY HH:MM:SS“.

`output : string * workflowname`
: Name als Referenz für den Workflow. Wird aus dem Namen aus der Workflow-Beschreibung und einer laufenden Nummer gebildet.

Beschreibung:

Die Funktion *InitWorkflow* startet einen neuen Workflow. Sie bekommt als Eingabe eine Datei mit der Workflow-Beschreibung (Parameter *workflowfile*) und eine Datums- und Zeitangabe (Parameter *endtime*), zu der das Ergebnis des Workflows bereitstehen muß.

Die Funktion lädt die Workflow-Beschreibung und übergibt sie dem Parser. Dieser liest die Beschreibung ein und speichert die Daten in einer Struktur. Anschließend wird der Abhängigkeitsgraph für den Workflow erstellt. Dieser wird direkt aus den Aktivitäten, sowie den Kontrollfluß- und Datenflußangaben der Workflow-Beschreibung abgeleitet. Im Falle variabler Parallelität dient die hier erstellte Aktivität als Schablone für die zur Laufzeit entstehenden Aktivitäteninstanzen. Nun kann das aktuelle Datum und die aktuelle Zeit ermittelt und im Abhängigkeitsgraph die logischen Zeitpunkte der Meilensteine in reale Zeitpunkte umgerechnet werden. Daraufhin kann die zeitliche Durchführbarkeit des Workflows überprüft werden. Dazu werden die realen Zeitangaben mit der angestrebten Endzeit verglichen. Ist die Endzeit nicht mehr erreichbar, wird der Workflow abgewiesen. Kann der Workflow ausgeführt werden, wird seine Struktur in die Workflow-Liste eingetragen. Aktivitäten, die sofort ausgeführt werden können, werden in die Arbeitslisten der dafür in Frage kommenden Benutzer eingetragen.

Als Ausgabe liefert die Funktion einen Namen (Parameter *workflowname*) als Referenz auf den Workflow. Darüber können andere Funktionen auf den Workflow Bezug nehmen.

4.3.2 CancelWorkflow

Beendet Workflow-Ausführung oder bricht sie ab.

Definition der Metaphase-Methode:

```
message WepRoot:w0CancelWorkflow
(
   input  : string    classname    ::
   input  : string    workflowname ::
   output : integer * errorcode    ::
   output : integer * mfail
)
```

Parameter:

`input : string workflowname`
Name des zu beendenden Workflows. Wird von der Funktion *InitWorkflow* erstellt.

Beschreibung:

Die Funktion *CancelWorkflow* beendet einen abgeschlossenen Workflow oder bricht einen laufenden Workflow ab. Sie bekommt als Eingabe den Namen (Parameter *workflowname*), der als Referenz auf den Workflow dient.

Ist der Workflow noch nicht ordnungsgemäß abgeschlossen, das heißt, ist er noch nicht am Ende seines Kontrollflusses angekommen, so wird er abgebrochen. Dazu werden alle Benutzer, die noch an einer Aktivität des Workflows arbeiten, über ihre Arbeitslisten über den Abbruch informiert. Wird aktuell keine Aktivität des Workflows bearbeitet, so kann der Workflow direkt beendet werden. Dazu werden alle Aktivitäten des Workflows, die sich noch in den Arbeitslisten der Benutzer befinden, von dort entfernt. Die Workflow-Struktur wird aus der Workflow-Liste ausgeklinkt. Danach kann die gesamte Struktur, inklusive Abhängigkeitsgraph, gelöscht werden.

4.3.3 RegisterUser

Meldet einen Benutzer am WEP-System an.

Definition der Metaphase-Methode:

```
message WepRoot:w0RegisterUser
(
   input  : string    classname ::
   input  : string    username  ::
   input  : string    password  ::
   input  : string    userrole  ::
   output : integer * errorcode ::
   output : integer * mfail
)
```

Parameter:

`input : string username`
: Beliebiger Name eines Benutzers, der angemeldet werden soll.

`input : string password`
: Passwort des Benutzers. Dient im Augenblick nur als Dummy und wird nicht überprüft.

`input : string userrole`
: Rollenzuordnung des Benutzers. Legt fest, welche Aktivitäten der Benutzer bearbeiten kann. Die für einen Workflow erlaubten Rollennamen sind in dessen Workflow-Beschreibung festgelegt.

Beschreibung:

Die Funktion *RegisterUser* meldet einen neuen Benutzer am System an. Zur Definition des Benutzers dient sein Name (Parameter *username*), sein Passwort (Parameter *password*) und seine Rollenzuordnung (Parameter *userrole*).

Die Funktion trägt einen neuen Benutzer in die Benutzerliste ein. Dabei muß zuerst überprüft werden, ob er schon in der Liste existiert. Ist dies nicht der Fall, wird er eingetragen und seine Arbeitsliste wird erstellt. Danach werden alle laufenden Workflows, die die Rollenzuordnung des Benutzers zulassen, überprüft, ob in ihrem Kontrollfluß Aktivitäten für diese Rollenzuordnung zur Bearbeitung anstehen. Existieren für die Rolle des Benutzers zugelassene Aktivitäten, so werden sie in dessen Arbeitsliste eingetragen.

4.3.4 CancelUser

Meldet einen Benutzer vom WEP-System ab.

Definition der Metaphase-Methode:

```
message WepRoot:w0CancelUser
(
   input  : string    classname ::
   input  : string    username  ::
   output : integer * errorcode ::
   output : integer * mfail
)
```

Parameter:

`input : string username`
: Name des Benutzers, der abgemeldet werden soll. Wird dem System durch die Funktion *RegisterUser* bekannt gemacht.

Beschreibung:

Die Funktion *CancelUser* meldet einen Benutzer vom System ab. Als Identifikation für den Benutzer bekommt sie dessen Benutzernamen (Parameter *username*) als Eingabe.

Ein Benutzer kann nur dann vom System abgemeldet werden, wenn er gerade keine Aktivitäten bearbeitet. In einem solchen Fall könnten, bezüglich nachfolgender Aktivitäten, komplexe Inkonsistenzprobleme auftreten. Die Funktion überprüft zuerst, ob der Benutzer überhaupt existiert. Danach kann getestet werden, ob in der Arbeitsliste des Benutzers aktive Aktivitäten eingetragen sind. Im zutreffenden Fall wird die Funktion abgebrochen. Anderenfalls werden alle Einträge der Arbeitsliste und die Arbeitsliste selber gelöscht und der Benutzer aus der Benutzerliste entfernt.

4.3.5 PollWorkList

Ermittelt den Inhalt der Arbeitsliste eines Benutzers.

Definition der Metaphase-Methode:

```
message WepRoot:w0PollWorkList
(
   input  : string         classname ::
   input  : string         username  ::
   output : SetOfObjects * worklist  ::
   output : integer *      errorcode ::
   output : integer *      mfail
)
```

Parameter:

`input : string username`
: Name des Benutzers, dessen Arbeitsliste ermittelt werden soll.

`output : SetOfObjects * worklist`
: Stellt die Arbeitsliste dar. Jeder Eintrag der Menge, also jedes Objekt, entspricht einem Eintrag beziehungsweise einer Aktivität in der Arbeitsliste.

Beschreibung:

Die Funktion *PollWorkList* liest die Arbeitsliste eines Benutzers aus. Der Benutzer wird per Eingabe (Parameter *username*) bestimmt.

Mit dieser Funktion bekommt der Benutzer Informationen über alle seine gestarteten Aktivitäten, sowie über die Aktivitäten, die noch von niemandem gestartet wurden, aber von seiner Rolle ausgeführt werden können. Hiermit kann also ständig der Zustand der Arbeitsliste gepollt werden. Die Funktion testet zuerst, ob der Benutzer existiert. Danach wird überprüft, ob die Aktivitäten, die in der Arbeitsliste des Benutzers stehen, noch im Zeitrahmen liegen. Ist dies nicht der Fall, wird in der Arbeitsliste ein entsprechendes Warn-Flag gesetzt. Nun erstellt die Funktion aus der Arbeitsliste eine Metaphase-Objektmenge. Ein Objekt dieser Menge entspricht dem Objekttyp WepWLObj und stellt einen Eintrag der Arbeitsliste dar.

Als Ausgabe liefert die Funktion einen Zeiger auf die Objektmenge (Parameter *worklist*).

4.3.6 StartActivity

Startet die Bearbeitung einer Aktivität.

Definition der Metaphase-Methode:

```
message WepRoot:w0StartActivity
(
   input  : string      classname    ::
   input  : string      username     ::
   input  : string      workflowname ::
   input  : string      activityname ::
   output : ObjectPtr * activityarea ::
   output : integer *   errorcode    ::
   output : integer *   mfail
)
```

Parameter:

`input : string username`
: Name des Benutzers, der die Aktivität bearbeiten will.

`input : string workflowname`
: Name des Workflows, zu dem die Aktivität gehört.

`input : string activityname`
: Name der Aktivität, die gestartet werden soll. Kann der Arbeitsliste entnommen werden.

`output : ObjectPtr * activityarea`
: Stellt eine Beschreibung der Aktivität dar. Enthält alle für den Benutzer notwendigen Daten, die die Eingabeobjekte, Ausgabeobjekte und Programmschritte betreffen.

Beschreibung:

Die Funktion *StartActivity* startet eine Aktivität aus der Arbeitsliste eines Benutzers. Sie bekommt zur Identifikation des Benutzers dessen Namen (Parameter *username*) als Eingabe. Zur Identifikation der Aktivität reichen der Name des Workflows (Parameter *workflowname*), sowie der Name der Aktivität (Parameter *activityname*). Damit kann die Aktivität eindeutig definiert werden.

Die Funktion überprüft die Existenz des Benutzers, ebenso die Existenz des Workflows und der Aktivität. Jetzt kann die Aktivität aus den Arbeitslisten aller anderen Benutzer, in denen sie eventuell noch eingetragen ist, ausgetragen werden. Dann wird aus den Aktivitätsdaten der Workflow-Beschreibung und des Abhängigkeitsgraphen das *activityarea*-Objekt erstellt. Es entspricht dem Objekttyp WepAAObj und enthält alle Informationen, die der Benutzer zur Bearbeitung der Aktivität benötigt. Darunter fällt die Beschreibung der Eingabeobjekte, der Meilensteine für die zu erstellenden Ausgabeobjekte und die verfügbaren Programmschritte

(Werkzeuge). Zuletzt wird der Status der Aktivität in der Arbeitsliste des Benutzers und im Abhängigkeitsgraph auf *running* gesetzt.

Als Ausgabe liefert die Funktion einen Zeiger auf das Objekt, das die Aktivitätendaten enthält (Parameter *activityarea*).

4.3.7 SuspendActivity

Unterbricht die Ausführung einer Aktivität für eine unbestimmte Zeit.

Definition der Metaphase-Methode:

```
message WepRoot:w0SuspendActivity
(
   input  : string    classname    ::
   input  : string    username     ::
   input  : string    workflowname ::
   input  : string    activityname ::
   input  : ObjectPtr activityarea ::
   output : integer * errorcode    ::
   output : integer * mfail
)
```

Parameter:

`input : string username`
: Name des Benutzers, der die Aktivität unterbrechen will.

`input : string workflowname`
: Name des Workflows, zu dem die Aktivität gehört.

`input : string activityname`
: Name der Aktivität, die unterbrochen werden soll.

`input : ObjectPtr activityarea`
: Stellt eine Beschreibung der Aktivität dar. Wird von der Funktion *StartActivity* erstellt. Enthält zusätzlich die vom Benutzer getätigten Arbeiten.

Beschreibung:

Die Funktion *SuspendActivity* unterbricht die Ausführung einer Aktivität für eine unbestimmte Zeit. Über die Eingaben der Funktion wird der Benutzer (Parameter *username*), sowie der Workflow (Parameter *workflowname*) und die Aktivität (Parameter *activityname*) identifiziert. Außerdem bekommt die Funktion das von der Funktion *StartActivity* erstellt activityarea-Objekt (Parameter *activityarea*). Es enthält jedoch zusätzliche Daten, die die bisherige Arbeit des Benutzers beschreiben. Dies betrifft den Stand der Ausgabeobjekte und die sich gerade in Arbeit befindenden Objekte.

Die Funktion überprüft die Existenz des Benutzers, des Workflows und der Aktivität. Danach wird das *activityarea*-Objekt zwischengespeichert. In der Arbeitsliste des Benutzers wird der Status der Aktivität auf *suspended* geändert.

Die Aktivität kann mit der Funktion *ResumeActivity* wieder fortgesetzt werden.

4.3.8 ResumeActivity

Setzt die unterbrochene Bearbeitung einer Aktivität wieder fort.

Definition der Metaphase-Methode:

```
message WepRoot:w0ResumeActivity
(
   input  : string      classname    ::
   input  : string      username     ::
   input  : string      workflowname ::
   input  : string      activityname ::
   output : ObjectPtr * activityarea ::
   output : integer *   errorcode    ::
   output : integer *   mfail
)
```

Parameter:

`input : string username`
: Name des Benutzers, der die Aktivität fortsetzen will.

`input : string workflowname`
: Name des Workflows, zu dem die Aktivität gehört.

`input : string activityname`
: Name der Aktivität, die fortgesetzt werden soll.

`output : ObjectPtr * activityarea`
: Stellt eine Beschreibung der Aktivität dar. Wird von der Funktion *StartActivity* erstellt und von der Funktion *SuspendActivity* beim Unterbrechen zwischengespeichert.

Beschreibung:

Die Funktion *ResumeActivity* setzt die mit der Funktion *SuspendActivity* unterbrochene Bearbeitung einer Aktivität wieder fort. Über die Eingaben der Funktion wird der Benutzer (Parameter *username*) sowie der Workflow (Parameter *workflowname*) und die Aktivität (Parameter *activityname*) identifiziert.

Die Funktion überprüft die Existenz des Benutzers, des Workflows und der Aktivität. In der Arbeitsliste des Benutzers wird der Status der Aktivität daraufhin von *suspended* wieder auf *running* geändert.

Als Ausgabe liefert die Funktion die bei *SuspendActivity* zwischengespeicherten Aktivitätsdaten als *activityarea*-Objekt. Es entspricht dem Objekttyp WepAAObj.

4.3.9 PrepareFinishActivity

Bereitet den Abschluß einer Aktivität vor und ermittelt deren mögliche Rückgabewerte.

Definition der Metaphase-Methode:

```
message WepRoot:w0PrepFinishActivity
(
   input  : string         classname    ::
   input  : string         username     ::
   input  : string         workflowname ::
   input  : string         activityname ::
   input  : ObjectPtr      activityarea ::
   output : SetOfStrings * returncodes  ::
   output : integer *      errorcode    ::
   output : integer *      mfail
)
```

Die Abkürzung des Funktionsnamens auf *w0PrepFinishActivity* ist durch Metaphase bedingt. Metaphase erlaubt nur eine maximale Länge von 22 Zeichen für einen Funktionsnamen.

Parameter:

`input : string username`
: Name des Benutzers, der die Aktivität abschließen will.

`input : string workflowname`
: Name des Workflows, zu dem die Aktivität gehört.

`input : string activityname`
: Name der Aktivität, die abgeschlossen werden soll.

`input : ObjectPtr activityarea`
: Stellt eine Beschreibung der Aktivität dar. Wird von der Funktion StartActivity erstellt. Enthält zusätzliche Informationen zu den vom Benutzer getätigten Arbeiten.

`output : SetOfStrings * returncodes`
: Entspricht einer Menge von Zeichenketten. Jede Zeichenkette benennt einen möglichen Rückgabewert der Aktivität.

Beschreibung:

Die Funktion *PrepareFinishActivity* bereitet den Abschluß einer Aktivität vor, indem sie deren mögliche Rückgabewerte ermittelt. Über die Eingaben der Funktion wird der Benutzer (Parameter *username*), der Workflow (Parameter *workflowname*) und die Aktivität (Parameter *activityname*) identifiziert. Über den Parameter *activityarea* werden außerdem die Daten der bearbeiteten Aktivität zurückgeliefert.

Die Funktion überprüft die Existenz des Benutzers, des Workflows und der Aktivität. In der Arbeitsliste des Benutzers wird der Status der Aktivität auf *finished* geändert und die Daten der Aktivität im Abhängigkeitsgraph werden, entsprechend des *activityarea*-Objekts, aktualisiert. Das *activityarea*-Objekt wird gelöscht, da es nicht mehr benötigt wird. Daraufhin werden die

Qualitätsstufen der Ausgabeobjekte mit den erforderlichen Qualitätsstufen der Rückgabewerte aus der Workflow-Beschreibung verglichen.

Die Rückgabewerte der erreichten Qualitätsstufen werden in einer Zeichenketten-Menge zusammengefaßt (Parameter *returncodes*) und als Ausgabe zurückgeliefert.

4.3.10 FinishActivity

Beendet die Ausführung einer Aktivität.

Definition der Metaphase-Methode:

```
message WepRoot:w0FinishActivity
(
   input  : string    classname      ::
   input  : string    username       ::
   input  : string    workflowname   ::
   input  : string    activityname   ::
   input  : string    returncodename ::
   output : integer * errorcode      ::
   output : integer * mfail
)
```

Parameter:

`input : string username`
: Name des Benutzers, der die Aktivität beenden will.

`input : string workflowname`
: Name des Workflows, zu dem die Aktivität gehört.

`input : string activityname`
: Name der Aktivität, die beendet werden soll.

`input : string returncodename`
: Name des vom Benutzer ausgewählten Rückgabewertes für die Aktivität. Die Liste möglicher Rückgabewerte wird von der Funktion *PrepareFinishActivity* erstellt.

Beschreibung:

Die Funktion *FinishActivity* beendet eine Aktivität. Vor dem Aufruf dieser Funktion muß bereits die Funktion *PrepareFinishActivity* aufgerufen worden sein, um die Beendigung der Aktivität vorzubereiten. Über die Eingaben der Funktion wird der Benutzer (Parameter *username*), sowie der Workflow (Parameter *workflowname*) und die Aktivität (Parameter *activityname*) identifiziert. Der Parameter *returncodename* übergibt den vom Benutzer ausgewählten Rückgabewert der Aktivität. Die Liste der möglichen Rückgabewerte wurde zuvor anhand der Qualitätsstufen der erstellten Ausgabeobjekte von der Funktion *PrepareFinishActivity* an den Benutzer geliefert.

Die Funktion überprüft die Existenz des Benutzers, des Workflows und der Aktivität. Dann kann die beendete Aktivität wird aus der Arbeitsliste des Benutzers entfernt und alle mit der Funktion *FetchObject* gesperrten Objekte wieder freigegeben werden. Arbeitet eine nachfolgende Aktivität auf einem von dieser Aktivität bisher gesperrten Objekt, so wird dessen Benutzer über seine Arbeitsliste mitgeteilt, daß er jetzt das Objekt exklusiv sperren kann. Im Abhängigkeitsgraph wird die Beendigung der Aktivität vermerkt.

4.3.11 FetchObject

Fordert den Zugriff auf ein Eingabeobjekt einer Aktivität an.

Definition der Metaphase-Methode:

```
message WepRoot:w0FetchObject
(
   input  : string    classname    ::
   input  : string    username     ::
   input  : string    workflowname ::
   input  : string    activityname ::
   input  : string    objectname   ::
   output : integer * dependent    ::
   output : integer * errorcode    ::
   output : integer * mfail
)
```

Parameter:

`input : string username`
: Name des Benutzers, der das Eingabeobjekt anfordert.

`input : string workflowname`
: Name des Workflows, zu dem die Aktivität gehört.

`input : string activityname`
: Name der Aktivität, zu der das Eingabeobjekt gehört.

`input : string objectname`
: Name des Eingabeobjekts. Wird dem Benutzer über die Aktivitätsdaten (*activityarea*) der Funktion *StartActivity* bekannt gemacht.

`ouput : integer * dependent`
: Gibt an, ob es sich bezüglich des betrachteten Objekts um eine abhängige (=1) oder unabhängige (=0) Aktivität handelt. Nur bei einer unabhängigen Aktivität kann eine neue offizielle Objektversion festgelegt werden.

Beschreibung:

Die Funktion *FetchObject* fordert den Zugriff auf ein Eingabeobjekt einer Aktivität an. Über die Eingaben werden der Funktion der Benutzer (Parameter *username*), der Workflow (Parameter

workflowname), die Aktivität (Parameter *activityname*) und das Objekt (Parameter *objectname*) bekanntgemacht.

Die Funktion überprüft die Existenz des Benutzers, des Workflows, der Aktivität und des Eingabeobjekts. Danach wird von der letzten offiziellen Version des Eingabeobjekts eine Kopie erzeugt. Im Abhängigkeitsgraph wird dies bei der Aktivität vermerkt. Alle zukünftigen Werkzeugaufrufe über die Funktion *StartProgramStep* arbeiten auf dieser Kopie. Handelt es sich bezüglich des betrachteten Objektes bei der Aktivität um eine unabhängige Aktivität, so ist nur diese in der Lage eine neue offizielle Objektversion zu erstellen. Eine abhängige Aktivität kann zwar auf ihrer Kopie ebenfalls Änderungen durchführen, sie kann diese aber nicht als offizielle Objektversion speichern.

Die Unterscheidung, ob die Aktivität abhängig oder unabhängig ist, wird über den Parameter *dependent* zurückgemeldet.

4.3.12 ReleaseObject

Gibt ein Ausgabeobjekt für nachfolgende Aktivitäten frei.

Definition der Metaphase-Methode:

```
message WepRoot:w0ReleaseObject
(
   input  : string    classname    ::
   input  : string    username     ::
   input  : string    workflowname ::
   input  : string    activityname ::
   input  : string    objectname   ::
   output : integer * errorcode    ::
   output : integer * mfail
)
```

Parameter:

`input : string username`
: Name des Benutzers, der das Ausgabeobjekt freigibt.

`input : string workflowname`
: Name des Workflows, zu dem die Aktivität gehört.

`input : string activityname`
: Name der Aktivität, zu der das Ausgabeobjekt gehört.

`input : string objectname`
: Name des Ausgabeobjekts. Wird dem Benutzer über die Aktivitätsdaten (*activityarea*) der Funktion *StartActivity* bekannt gemacht.

Beschreibung:

Die Funktion *ReleaseObject* gibt eine vorzeitige oder fertige Version eines Ausgabeobjekts für die Bearbeitung durch nachfolgende Aktivitäten frei. Über die Eingaben der Funktion werden der Benutzer (Parameter *username*), der Workflow (Parameter *workflowname*), die Aktivität (Parameter *activityname*) und das Objekt (Parameter *objectname*) identifiziert.

Die Funktion überprüft die Existenz des Benutzers, des Workflows, der Aktivität und des Ausgabeobjekts. Dann wird die aktuelle Qualitätsstufe des Objekts mit den Qualitätsanforderungen der nachfolgenden Aktivitäten verglichen. Sind die Anforderungen erfüllt und eine betroffene Aktivität wurde noch nicht gestartet, so wird sie nun in den Arbeitslisten der dafür zuständigen Benutzer eingetragen. Handelt es sich um Aktivitäten, die bereits aktiv sind, wird in der Arbeitsliste des Benutzers dieser Aktivität eingetragen, daß eine neue Objektversion verfügbar ist. Wurde für die Weitergabe dieses Ausgabeobjekts in der Workflow-Beschreibung der *Concurrent Mode* CONSOLIDATION spezifiziert, so startet das System mittels der Funktion *StartConsolidation* automatisch eine Konsolidierungsrunde. Nachfolgende Aktivitäten können über den Abhängigkeitsgraph ermittelt werden. Im Abhängigkeitsgraph werden auch die Abhängigkeiten bezüglich vorzeitig weitergegebener Objektversionen vermerkt.

4.3.13 RevokeObject

Nimmt ein freigegebenes Objekt wieder zurück.

Definition der Metaphase-Methode:

```
message WepRoot:w0RevokeObject
(
   input  : string    classname    ::
   input  : string    username     ::
   input  : string    workflowname ::
   input  : string    activityname ::
   input  : string    objectname   ::
   output : integer * errorcode    ::
   output : integer * mfail
)
```

Parameter:

`input : string username`
: Name des Benutzers, der das Ausgabeobjekt freigegeben hat und nun wieder zurücknimmt.

`input : string workflowname`
: Name des Workflows, zu dem die Aktivität gehört.

`input : string activityname`
: Name der Aktivität, zu der das Ausgabeobjekt gehört.

`input : string objectname`
Name des Ausgabeobjekts. Wird dem Benutzer über die Aktivitätsdaten (*activityarea*) der Funktion *StartActivity* bekannt gemacht.

Beschreibung:

Die Funktion *RevokeObject* nimmt ein ursprünglich bereits freigegebenes Ausgabeobjekt wieder in die Bearbeitung zurück und entzieht es damit den nachfolgenden Aktivitäten. Über die Eingaben der Funktion werden der Benutzer (Parameter *username*), der Workflow (Parameter *workflowname*), die Aktivität (Parameter *activityname*) und das Objekt (Parameter *objectname*) identifiziert.

Die Funktion überprüft die Existenz des Benutzers, des Workflows, der Aktivität und des Ausgabeobjekts. In den Arbeitslisten der Benutzer nachfolgender Aktivitäten wird die Rücknahme durch das Setzen eines Flags kenntlich gemacht. Sie dürfen mit der Objektversion nicht mehr weiterarbeiten. Außerdem werden nun nicht mehr vorhandene Abhängigkeiten im Abhängigkeitsgraph gelöscht.

4.3.14 RequestObject

Fordert eine neue Objektversion an.

Definition der Metaphase-Methode:

```
message WepRoot:w0RequestObject
(
   input  : string    classname    ::
   input  : string    username     ::
   input  : string    workflowname ::
   input  : string    activityname ::
   input  : string    objectname   ::
   output : integer * errorcode    ::
   output : integer * mfail
)
```

Parameter:

`input : string username`
Name des Benutzers, der eine neue Version eines Eingabeobjekts anfordert.

`input : string workflowname`
Name des Workflows, zu dem die Aktivität gehört.

`input : string activityname`
Name der Aktivität, zu der das Eingabeobjekt gehört.

`input : string objectname`
Name des Eingabeobjekts. Wird dem Benutzer über die Aktivitätsdaten (*activityarea*) der Funktion *StartActivity* bekannt gemacht.

Beschreibung:

Die Funktion *RequestObject* fordert von einem vorzeitig erhaltenen Eingabeobjekt eine neue Version an. Über die Eingaben der Funktion werden der Benutzer (Parameter *username*), der Workflow (Parameter *workflowname*), die Aktivität (Parameter *activityname*) und das Objekt (Parameter *objectname*) identifiziert.

Die Funktion überprüft die Existenz des Benutzers, des Workflows, der Aktivität und des Eingabeobjekts. Danach wird über den Abhängigkeitsgraphen die Vorgänger-Aktivität ermittelt. In der Arbeitsliste des Benutzers dieser Aktivität kann nun durch das Setzen eines Flags kenntlich gemacht werden, daß eine neue Objektversion benötigt wird.

4.3.15 StartProgramStep

Startet die Bearbeitung eines Objekts mit einem Werkzeug.

Definition der Metaphase-Methode:

```
message WepRoot:w0StartProgramStep
(
   input  : string    classname    ::
   input  : string    username     ::
   input  : string    workflowname ::
   input  : string    activityname ::
   input  : string    objectname   ::
   input  : string    toolname     ::
   output : integer * errorcode    ::
   output : integer * mfail
)
```

Parameter:

`input : string username`
: Name des Benutzers, der ein Werkzeug starten will.

`input : string workflowname`
: Name des Workflows, zu dem die Aktivität gehört.

`input : string activityname`
: Name der Aktivität, zu der das Objekt gehört.

`input : string objectname`
: Name des Objekts, das mittels des Werkzeugs bearbeitet werden soll.

`input : string toolname`
: Name des Werkzeugs (Programmschrittes). Wird dem Benutzer über die Aktivitätsdaten (*activityarea*) der Funktion *StartActivity* bekannt gemacht.

Beschreibung:

Die Funktion *StartProgramStep* startet die Ausführung eines Programmschrittes, beziehungsweise die Bearbeitung eines Objekts mit einem Werkzeug. Über die Eingaben der Funktion werden der Benutzer (Parameter *username*), der Workflow (Parameter *workflowname*), die Aktivität (Parameter *activityname*), das Objekt (Parameter *objectname*) und das Werkzeug (der Programmschritt, Parameter *toolname*) identifiziert.

Die Funktion überprüft die Existenz des Benutzers, des Workflows, der Aktivität, des Objekts und die Verfügbarkeit des Programmschrittes innerhalb der Aktivität. Die möglichen Programmschritte einer Aktivität, werden in der Workflow-Beschreibung spezifiziert. Die Ausführung eines Werkzeuges, das den Programmschritt darstellt, verläuft asynchron. Mit der Funktion *FinishProgramStep* kann ein Programmschritt vor seinem regulären Ende abgebrochen werden. Nach seinem Ende kann damit das Ergebnis ermittelt werden. Mit der Funktion *PollProgramStep* kann festgestellt werden, ob die Ausführung des Werkzeuges noch läuft oder bereits beendet ist. Die Ausführung eines Programmschrittes wird im Abhängigkeitsgraph vermerkt.

4.3.16 PollProgramStep

Ermittelt Status einer Werkzeugausführung.

Definition der Metaphase-Methode:

```
message WepRoot:w0PollProgramStep
(
   input  : string    classname    ::
   input  : string    username     ::
   input  : string    workflowname ::
   input  : string    activityname ::
   input  : string    toolname     ::
   output : integer * finished     ::
   output : integer * errorcode    ::
   output : integer * mfail
)
```

Parameter:

`input : string username`
Name des Benutzers, der Informationen über den Ablauf eines Werkzeugs einholen will.

`input : string workflowname`
Name des Workflows, zu dem die Aktivität gehört.

`input : string activityname`
Name der Aktivität, innerhalb der das Werkzeug gestartet wurde.

`input : string toolname`
Name des Werkzeugs (Programmschrittes), das ausgeführt wurde.

`output : integer * finished`
Gibt zurück, ob die Werkzeugausführung beendet ist (=1) oder noch läuft (=0).

Beschreibung:

Die Funktion *PollProgramStep* ermittelt den Status einer Werkzeugausführung. Das Werkzeug wurde mit der Funktion *StartProgramStep* gestartet. Über die Eingaben der Funktion werden der Benutzer (Parameter *username*), der Workflow (Parameter *workflowname*), die Aktivität (Parameter *activityname*) und das Werkzeug (Parameter *toolname*) identifiziert.

Die Funktion überprüft die Existenz des Benutzers, des Workflows, der Aktivität und ob das Werkzeug überhaupt gestartet wurde. Ob die Werkzeugausführung noch aktiv ist oder bereits beendet wurde, wird über den Parameter *finished* zurückgemeldet. Ist die Werkzeugausführung beendet, kann mit der Funktion *FinishProgramStep* das Ergebnis ermittelt werden.

4.3.17 FinishProgramStep

Beendet die Ausführung eines Werkzeugs.

Definition der Metaphase-Methode:

```
message WepRoot:w0FinishProgramStep
(
   input  : string    classname    ::
   input  : string    username     ::
   input  : string    workflowname ::
   input  : string    activityname ::
   input  : string    toolname     ::
   output : string *  toolresult   ::
   output : integer * errorcode    ::
   output : integer * mfail
)
```

Parameter:

`input : string username`
Name des Benutzers, der die Werkzeugausführung beenden will.

`input : string workflowname`
Name des Workflows, zu dem die Aktivität gehört.

`input : string activityname`
Name der Aktivität, innerhalb der das Werkzeug gestartet wurde.

`input : string toolname`
Name des Werkzeugs (Programmschrittes), das ausgeführt wurde.

`output : string * toolresult`
Gibt das Ergebnis der Werkzeugausführung zurück. Numerische Werte werden in eine Zeichenkette konvertiert.

Beschreibung:

Die Funktion *FinishProgramStep* beendet die Ausführung eines Werkzeugs und ermittelt das Ergebnis. Das Werkzeug wurde mit der Funktion *StartProgramStep* gestartet. Über die Eingaben der Funktion werden der Benutzer (Parameter *username*), der Workflow (Parameter *workflowname*), die Aktivität (Parameter *activityname*) und das Werkzeug (Parameter *toolname*) identifiziert.

Die Funktion überprüft die Existenz des Benutzers, des Workflows, der Aktivität und ob das Werkzeug überhaupt gestartet wurde. Ob die Werkzeugausführung noch aktiv ist oder bereits beendet wurde, kann über die Funktion *PollProgramStep* ermittelt werden. Ist die Werkzeugausführung noch aktiv, so wird sie abgebrochen. Im Abhängigkeitsgraph wird das Ende des Werkzeugausführung vermerkt.

Über den Parameter *toolresult* wird das Ergebnis einer fehlerfreien oder einer abgebrochenen Werkzeugausführung zurückgeliefert. Weitere Ergebnisse werden direkt in der mit der Funktion *StartProgramStep* übergebenen Objektkopie eingetragen.

4.3.18 StartConsolidation

Beruft eine Konsolidierungsrunde ein.

Definition der Metaphase-Methode:

```
message WepRoot:w0StartConsolidation
(
   input  : string    classname       ::
   input  : string    username        ::
   input  : string    workflowname    ::
   input  : string    activityname    ::
   input  : string    objectname      ::
   output : integer * consolidationid ::
   output : integer * errorcode       ::
   output : integer * mfail
)
```

Parameter:

`input : string username`
: Name des Benutzers, der die Konsolidierungsrunde startet.

`input : string workflowname`
: Name des Workflows, zu dem die Aktivität gehört.

`input : string activityname`
: Name der Aktivität, zu der das Objekt gehört.

`input : string objectname`
: Name des Objekts, über das in der Konsolidierungsrunde abgestimmt werden soll.

`output : integer * consolidationid`
Gibt eine Referenz auf die Konsolidierungsrunde zurück, über die sie später direkt angesprochen werden kann. Entspricht einer einfachen Sequenznummer.

Beschreibung:

Die Funktion *StartConsolidation* beruft eine Konsolidierungsrunde ein. Über die Eingaben der Funktion werden der Benutzer (Parameter *username*), der Workflow (Parameter *workflowname*), die Aktivität (Parameter *activityname*) und das Objekt (Parameter *objectname*), über das abgestimmt werden soll, identifiziert.

Die Funktion überprüft die Existenz des Benutzers, des Workflows, der Aktivität und des Objekts. Dann werden über den Abhängigkeitsgraph zuerst alle Aktivitäten beziehungsweise deren Benutzer ermittelt, die mit einer Version des Objekts arbeiten. Durch das Setzen eines Flags in den Arbeitslisten dieser Benutzer wird die Konsolidierungsrunde einberufen. Durch das Eintragen des Objektnamens in den Arbeitslisten, wird auf das Objekt verwiesen, über das abgestimmt werden soll.

Über den Parameter *consolidationid* wird eine Sequenznummer zurückgegeben. Sie dient für alle nachfolgenden Funktionen der Konsolidierungsrunde als Referenz.

4.3.19 ProposeNewObject

Schlägt innerhalb einer Konsolidierungsrunde eine Objektversion vor.

Definition der Metaphase-Methode:

```
message WepRoot:w0ProposeNewObject
(
   input  : string    classname       ::
   input  : string    username        ::
   input  : integer   consolidationid ::
   input  : string    timeout         ::
   output : integer * errorcode       ::
   output : integer * mfail
)
```

Parameter:

`input : string username`
Name des Benutzers, der die Objektversion vorschlägt.

`input : integer consolidationid`
Referenz auf die Konsolidierungsrunde. Wird von der Funktion *StartConsolidation* erzeugt.

`input : string timeout`

Zeitdauer, innerhalb der alle Beteiligten abgestimmt haben müssen, bevor die Konsolidierungsrunde automatisch beendet wird. Im Format „DD.MM.YYYY HH:MM:SS“.

Beschreibung:

Die Funktion *ProposeNewObject* kann nur innerhalb einer Konsolidierungsrunde aufgerufen werden. Sie schlägt eine neue Objektversion vor. Über die Eingaben der Funktion werden der Benutzer (Parameter *username*) und die Konsolidierungsrunde (Parameter *consolidationid*) identifiziert. Der Parameter *timeout* legt die Zeitspanne fest, innerhalb der gültige Stimmen abgegeben werden können.

Die Funktion überprüft die Existenz des Benutzers und der Konsolidierungsrunde. Die aktuelle Objektversion der betroffenen Aktivität dieses Benutzers wird mit dieser Funktion zur Abstimmung vorgeschlagen. Die Objektversion wird in der Arbeitsliste eingetragen, um jeden Beteiligten darüber zu informieren. Alle in der Konsolidierungsrunde vorhandenen Benutzer können nun mit den Funktionen *AgreeNewObject* oder *RejectNewObject* ihre Meinung über die Objektversion äußern.

4.3.20 AgreeNewObject

Stimmt innerhalb einer Konsolidierungsrunde der vorgeschlagenen Objektversion zu.

Definition der Metaphase-Methode:

```
message WepRoot:w0AgreeNewObject
(
   input  : string    classname       ::
   input  : string    username        ::
   input  : integer   consolidationid ::
   output : integer * errorcode       ::
   output : integer * mfail
)
```

Parameter:

`input : string username`
Name des Benutzers, der der Objektversion zustimmt.

`input : integer consolidationid`
Referenz auf die Konsolidierungsrunde. Wird von der Funktion *StartConsolidation* erzeugt.

Beschreibung:

Die Funktion *AgreeNewObject* kann nur innerhalb einer Konsolidierungsrunde aufgerufen werden. Sie stimmt einer vorgeschlagenen Objektversion zu. Über die Eingaben der Funktion werden der Benutzer (Parameter *username*) und die Konsolidierungsrunde (Parameter *consolidationid*) identifiziert.

Die Funktion überprüft die Existenz des Benutzers und der Konsolidierungsrunde. Der Benutzer stimmt der zuvor mit der Funktion *ProposeNewObject* vorgeschlagenen Objektversion zu. Dies wird intern gespeichert und am Ende der Konsolidierungsrunde mit der Funktion *EndConsolidation* dem Benutzer, der die Objektversion vorgeschlagen hat, überliefert.

4.3.21 RejectNewObject

Lehnt innerhalb einer Konsolidierungsrunde die vorgeschlagene Objektversion ab.

Definition der Metaphase-Methode:

```
message WepRoot:w0RejectNewObject
(
   input  : string    classname       ::
   input  : string    username        ::
   input  : integer   consolidationid ::
   output : integer * errorcode       ::
   output : integer * mfail
)
```

Parameter:

`input : string username`
: Name des Benutzers, der die Objektversion ablehnt.

`input : integer consolidationid`
: Referenz auf die Konsolidierungsrunde. Wird von der Funktion *StartConsolidation* erzeugt.

Beschreibung:

Die Funktion *RejectNewObject* kann nur innerhalb einer Konsolidierungsrunde aufgerufen werden. Sie lehnt eine vorgeschlagene Objektversion ab. Über die Eingaben der Funktion werden der Benutzer (Parameter *username*) und die Konsolidierungsrunde (Parameter *consolidationid*) identifiziert.

Die Funktion überprüft die Existenz des Benutzers und der Konsolidierungsrunde. Der Benutzer lehnt die zuvor mit der Funktion *ProposeNewObject* vorgeschlagene Objektversion ab. Dies wird intern gespeichert und am Ende der Konsolidierungsrunde mit der Funktion *EndConsolidation* dem Benutzer, der die Objektversion vorgeschlagen hat, überliefert.

4.3.22 EndConsolidation

Beendet eine Konsolidierungsrunde.

Definition der Metaphase-Methode:

```
message WepRoot:w0EndConsolidation
(
   input  : string    classname       ::
   input  : string    username        ::
   input  : integer   consolidationid ::
   output : integer * difference      ::
   output : integer * errorcode       ::
   output : integer * mfail
)
```

Parameter:

`input : string username`
: Name des Benutzers, der die Konsolidierungsrunde beendet. Ist im Normalfall derselbe, der sie gestartet hat.

`input : integer consolidationid`
: Referenz auf die Konsolidierungsrunde. Wird von der Funktion *StartConsolidation* erzeugt.

`output : integer * difference`
: Liefert die Differenz zwischen den zustimmenden und ablehnenden Entscheidungen bezüglich der vorgeschlagenen Objektversion. Ein positiver Wert bedeutet mehr Zustimmungen, ein negativer Wert bedeutet mehr Ablehnungen.

Beschreibung:

Die Funktion *EndConsolidation* beendet eine Konsolidierungsrunde. Über die Eingaben der Funktion werden der Benutzer (Parameter *username*) und die Konsolidierungsrunde (Parameter *consolidationid*) identifiziert.

Die Funktion überprüft die Existenz des Benutzers und der Konsolidierungsrunde. Durch das Setzen eines Flags in den Arbeitslisten der an der Konsolidierungsrunde beteiligten Benutzer, wird die Konsolidierungsrunde beendet. Danach werden die zustimmenden und ablehnenden Meinungen gegeneinander aufgerechnet und die Differenz über den Parameter *difference* zurückgegeben.

4.4 Stand der Implementierung

Die Prototyp-Implementierung konnte wegen der begrenzten Dauer der Diplomarbeit und dem erhöhten Aufwand durch die zusätzlichen Interaktionsformen und die komplexen Überprüfungen der Objektversionen und Meilensteine nicht vollendet werden. Vollständig implementiert sind das Parser-Modul, die Funktions-Schnittstelle und die Benutzer- und Arbeitslisten-Verwaltung.

Die Benutzer-Verwaltung wird bisher von der Implementierung noch selber kontrolliert und nicht auf die entsprechenden Metaphase-Funktionen abgebildet. Dies ist für einen Prototyp ausreichend. Sobald die Rollenzuordnung im WEP-Modell umfassend definiert ist, kann hier eine Anpassung an Metaphase erfolgen. Der augenblickliche Stand der Benutzer-Verwaltung

impliziert außerdem den Nachteil, daß einem Benutzer nur dann Aktivitäten in die Arbeitsliste eingetragen werden, wenn er sich am WEP-System angemeldet hat (Funktion *RegisterUser*). Eine konkrete Implementierung erfordert hier natürlich eine Trennung zwischen dem Registrieren eines Benutzers und dem Anmelden eines Benutzers. Beispielsweise durch die zwei Funktionen *RegisterUser* und *LoginUser*. Dies gilt simultan auch für das Abmelden eines Benutzers.

Die Module der Workflow-Verwaltung und der Objekt-Verwaltung sind bisher nur teilweise implementiert. So ist es im Augenblick noch nicht möglich, Aktivitäten zu starten und zu kontrollieren. Die Implementierung wird jedoch in den folgenden Wochen fortgesetzt.

Eine weitere Diplomarbeit, zur Implementierung eines Klienten auf der E!Vista-Komponente, wurde bereits begonnen. Außerdem ist eine Diplomarbeit geplant, die eine grafische Modellierungskomponente entwickelt.

5 Zusammenfassung und Ausblick

Eine Produktentwicklung kann heutzutage durch die wachsende Komplexität der Produkte nicht mehr von einer einzelnen Person betreut werden. Eine Gruppe von Personen erleichtert und beschleunigt den Entwicklungsvorgang. Die Gruppenarbeit und die Produktentwicklung sind zwei Bereiche, die verschiedene Ansatzpunkte bieten, an denen Computersysteme unterstützend einwirken können. Klassische Informationssysteme eignen sich jedoch meistens nur unzureichend. Die Hauptprobleme dabei sind die dynamischen Aspekte, wie das Zuteilen neuer Aufträge, der Informations- und Datenaustausch zwischen den Gruppenmitgliedern oder die Festlegung und Überwachung zeitlicher Beschränkungen und Fristen. Verschiedene Ansätze und Ideen aus dem Forschungsgebiet des CSCW (Computer Supported Cooperative Work) erlauben eine elegantere Unterstützung dynamischer Abläufe, als dies mit herkömmlichen Ansätzen möglich wäre.

Die interessantesten Ansätze stammen sicherlich aus den Bereichen Workflow-Management, Projekt-Management und Groupware. Das Ziel von Groupware ist die kommunikationstechnologische Unterstützung einer Gruppe, die in Eigenregie unstrukturierte Aufgaben bearbeitet. Projekt-Management ist das Zerlegen eines Projekts in hantierbare Portionen und das Einbringen einer Struktur mittels Zeit- und / oder Kostenmanagement. Der Kern von Workflow-Management ist die Koordination und Kooperation strukturierter Arbeitsabläufe. Dabei bietet die hier grundsätzliche Trennung von Ablauflogik und dem eigentlichen Anwendungscode einen vielversprechenden Ausgangspunkt.

Für den Einsatz in der Produktentwicklung sind natürlich nicht alle Systeme, die diese Ansätze verfolgen, verwendbar, da spezielle Anforderungen erfüllt werden müssen. Produktentwicklungsprozesse sind im allgemeinen durch eine zunehmende Konkretisierung von der Idee zum fertigen Produkt und die Aufgliederung in Teilaufgaben aufgrund der Komplexität der Entwicklungsaufgabe charakterisiert. Ein System, das für die Produktentwicklung gut geeignet ist, sollte also zumindest eine Unterstützung bieten für

- flexible Abläufe innerhalb des Prozesses wegen der Aufgliederung in Teilaufgaben,
- unstrukturierte Teilprozesse zur Entfaltung der Kreativität der Entwickler,
- simultanes Bearbeiten sequentieller Prozeßschritte für kürzere Entwicklungszeiten,
- komplex strukturierte Daten aus der Entwicklung und
- ein Zeitmanagement zur Einhaltung vorgegebener Fristen.

Im konzeptionellen Teil dieser Diplomarbeit mußte das WEP-Workflow-Management-System auf der Basis einer Workflow-Beschreibungssprache weiterentwickelt werden. Das Ziel des WEP-Workflow-Management-Systems ist die aktive Unterstützung von Produktentwicklungsprozessen. Die Grundidee ist die Erweiterung eines prozeßorientierten Modells mit zielorientierten Aktivitäten. In den Aktivitäten eines WEP-Systems werden Programmschritte zusammengefaßt, ohne eine Reihenfolge zu modellieren. Die Ziele der Aktivitäten werden durch Meilensteine spezifiziert. Ein Meilenstein gibt an, zu welchem Zeitpunkt ein spezielles Ausgabeobjekt bereit stehen muß, um das Ziel der Aktivität zu erreichen. Durch das zusätzliche Einbeziehen von Datenqualitäten in ein objektorientiertes Datenmodell, wird die vorzeitige Datenweitergabe ermöglicht.

Die entwickelte Workflow-Beschreibungssprache enthält Konstrukte zur Darstellung des Kontrollflusses, des Datenflusses inklusive globaler Datenobjekte, der zielorientierten Aktivitäten mit ihren Meilensteinen, der verfügbaren Programmschritte (Werkzeuge) und der Objektklassen eines objektorientierten Datenmodells inklusive Qualitätsstufen. Über diese Konstrukte hinaus gibt es weitere Aspekte, die bisher recht vereinfacht verwendet wurden, aber zukünftig mehr

Aufmerksamkeit finden. Dies betrifft speziell das Zeitmanagement und die Rollenzuordnung. Für die Diplomarbeit wurden diese Bereiche ausgeklammert.

Anhand eines zwar vereinfachten aber dennoch realitätsnahen Beispiels eines typischen Produktentwicklungsprozesses wurde die Zweckmäßigkeit der Workflow-Beschreibungssprache und die Anwendung der Mechanismen des WEP-Modells überprüft und für korrekt befunden. Allerdings entstehen aus den Mechanismen des WEP-Modells auch Nachteile. Dies ist vor allem auf die große Komplexität der Workflowverwaltung zurückzuführen. Sie impliziert einen wesentlich höheren Aufwand, sowohl bei der Implementierung, als auch für das System zur Laufzeit.

Um die Mechanismen des WEP-Modells mit anderen Ansätzen zu vergleichen, wurden einige verwandte Modelle aus den Bereichen Workflow-Management, Projekt-Management und Groupware ausgewählt. Ihnen ist gemein, daß sie versuchen, die Planung, Verwaltung und Kooperation bei komplexen Arbeitsabläufen flexibler zu unterstützen und zu integrieren, als dies bisher der Fall war. Zuerst wurden die von ihnen verwendeten Mechanismen detaillierter betrachtet. Dann wurde versucht, das bereits für WEP verwendete Beispiel, in diesen Modellen darzustellen und den Ansatz gegen WEP abzugrenzen.

Bei der Beschreibung der Modelle und der Ausarbeitung der Beispiele wurde schnell klar, daß nur das WEP-Workflow-Management-System alle gestellten Anforderungen erfüllen kann. Die anderen Systeme entwickeln durchaus interessante Konzepte, entsprechen den Anforderungen aber immer nur teilweise. Die betrachteten Ansätze sind natürlich nur eine kleine Auswahl. Da die dynamischen Aspekte und die Flexibilität von Workflow-Management-Systemen und Projekt-Management-Systemen ein aktuelles Forschungsgebiet sind, werden ständig neue Konzepte vorgeschlagen.

Im Implementierungsteil der Arbeit wurde die Serverkomponente (WEP-Workflow-Engine) des WEP-Workflow-Management-Systems konzipiert und prototypisch implementiert. Der Server wurde auf Metaphase aufgesetzt. Metaphase ist ein umfassendes Informations-Management-System und bietet, aufbauend auf einem normalen Datenbanksystem, ein objektorientiertes Daten- und Versionsmanagement. Darüberhinaus bietet es eine verteilte Client- / Server-Architektur, die eine weite Palette von Plattformen unterstützt.

Die WEP-Serverkomponente wurde als Metaphase-Methodenserver implementiert und wird so direkt in die Metaphase-Serverkomponente eingebettet. Durch diese Methodik kann von jedem Metaphase-Klienten auf die neue Funktionalität des WEP-Servers zugegriffen werden. Der Zugriff auf die Mechanismen von WEP läuft dabei über die Funktionsschnittstelle der WEP-Serverkomponente. Um mittels einer Workflow-Beschreibung einen Workflow überhaupt starten zu können, war außerdem die Implementierung eines Parsermoduls in die Serverkomponente erforderlich.

Ein WEP-Workflow-Management-System besteht aus der Modellierungskomponente und der Laufzeitkomponente. Die Laufzeitkomponente teilt sich in den Server und den Klienten auf. Für ein vollständiges System müssen also noch ein Klient und eine Modellierungs-komponente, die die Workflow-Beschreibung erstellt, entworfen werden.

Literaturverzeichnis

[BDS98] T. Beuter, P. Dadam, P. Schneider, „The WEP Model: Adequate Workflow-Management for Engineering Processes“, Proc. 5th European Concurrent Engineering Conference, Universität Erlangen-Nürnberg, 1998

[Burg95] M. Burghardt, Siemens AG (Hrsg.), „Einführung in Projektmanagement“, Publicis MCD Verlag, Erlangen 1995

[Dada96] P. Dadam, „Verteilte Datenbanken und Client- / Server-Systeme“, Springer-Verlag, Berlin Heidelberg, 1996

[Dada98] P. Dadam, „Kooperative Informationssysteme und Workflow-Management“, Vorlesungsskript, Fakultät Informatik, Universität Ulm, Sommer 1998

[Ditt95] E. Dittrich, „Entwurf und Implementierung von Services zur Bereitstellung von Organisations- und Personalstrukturen für ein Workflow-Management-System“, Diplomarbeit, Institut Betriebssysteme, Technische Universität Dresden, 1995

[DKR+95] P. Dadam, K. Kuhn, M. Reichert, T. Beuter, M. Nathe, „ADEPT: Ein integrierender Ansatz zur Entwicklung flexibler, zuverlässiger kooperierender Assistenzsysteme in klinischen Anwendungsumgebungen“, Proc. GI-SI Jahrestagung, Zürich 1995

[Doyl79] J. Doyle, „A Truth Maintenance System“, Artificial Intelligence Band 12, Seiten 231-272, 1979

[EGR91] C. A. Ellis, S. J. Gibbs, G. L. Rein, „Groupware - Some Issues and Experiences“, Communications of the ACM, Volume 34, 1991

[ElGi89] C. A. Ellis, S. J. Gibbs, „Concurrency Control in Groupware Systems“, Proc. of the 1989 ACM SIGMOD, Portland 1989

[Frau98] Fraunhofer Institut - Software und Systemtechnik, „Was ist Workflow Management / Groupware ?“, http://www.do.isst.fhg.de/workflow/pages/Begriffe_Deutsch.html

[GaSa87] H. Garcia-Molina, K. Salem, „Sagas“, SIGMOD87, San Francisco 1987

[GGK+91] H. Garcia-Molina, D. Gawlick, J. Klein, K. Kleissner, K. Salem, „Modeling Long-Running Activities as Nested Sagas“, IEEE Data Engineering Bulletin, März 1991

[Gold96] S. Goldmann, „Procura: A Project Management Model of Concurrent Planning and Design“, Proc. of WET ICE ’96, IEEE press, 1996

[Grim97] M. Grimm, „ADEPT-TIME: Temporale Aspekte in flexiblen Workflow-Management-Systemen“, Diplomarbeit, Fakultät für Informatik, Universität Ulm, 1997

[GrRe93] J. Gray, A. Reuter, „Transaction Processing: Concepts and Techniques“, Morgan Kaufmann, 1993

[Herb96] J. Herbst, „Workflow-Management-Systeme: Konzepte und Einsatzmöglichkeiten“, Technischer Bericht F3-96-020, Daimler-Benz AG, 1996

[HJKW96] P. Heimann, G. Joeris, C. Krapp, B. Westfechtel, „DYNAMITE: Dynamic Task Nets for Software Process Management“, Proc. 18th Int. Conf. on Software Engineering, IEEE Computer Society Press, Berlin, März 1996

[HJKW97] P. Heimann, G. Joeris, C. Krapp, B. Westfechtel, „Graph-Based Software Process Management“, Int. Journal of Software Engineering and Knowledge Engineering, Vol. 7, World Scientific Publishing Company, 1997

[HKS94] U. Hasenkamp, S. Kirn, M. Syring, „CSCW - Computer Supported Cooperative Work - Informationssysteme für dezentralisierte Unternehmensstrukturen“, Addison-Wesley, Bonn 1994

[HRS98] J. Hagemeyer, R. Rolles, Y. Schmidt, „Defizite der Arbeitsverteilung in Workflow-Management-Systemen: Eine kritische Analyse“, D-CSCW ’98. B. G. Teubner, Stuttgart 1998

[Jabl95a] S. Jablonski, „Workflow-Management-Systeme: Motivation, Modellierung, Architektur“, Informatik Spektrum 18, Springer-Verlag 1995

[Jabl95b] S. Jablonski, „Workflow-Management-Systeme: Modellierung und Architektur“, International Thomson Publishing, Bonn 1995

[Joer96] G. Joeris, „Ausführungssemantik und Dynamikaspekte in einem netzbasierten Prozeßmodell“, Diplomarbeit, Informatik III, RWTH Aachen, 1996

[Joer97] G. Joeris, „Characterization of Integrated Process and Product Management“, Proc. of the Workshop ‘Arbeitsplatzrechner-Integration zur Prozeßverbesserung’, GI Jahrestagung ’97, Aachen, September 1997

[Joer98] G. Joeris, „Aspekte und Konzepte der Flexibilität in Workflow-Managementsystemen“, Proc. D-CSCW 98, Technical Report Angewandte Mathematik und Informatik, Universität Münster, 1998

[Jung97] V. Jungbluth, „Teamwork - Einführung in die EDV-gestützte Projektplanung“, c’t 7/97, Heinz Heise Verlag, Hannover 1997

[Jung98] V. Jungbluth, „Perfekt geplant - Projektmanagementsysteme im Vergleich“, c’t 4/98, Heinz Heise Verlag, Hannover 1998

[Kubi98] M. Kubicek, „Organisatorische Aspekte in flexiblen Workflow-Management-Systemen“, Diplomarbeit, Fakultät für Informatik, Universität Ulm, 1998

[Kupp93] H. Kupper, „Zur Kunst der Projektsteuerung: Qualifikation und Aufgaben eines Projektleiters“, R. Oldenbourg Verlag, München 1993

[LoSc93] P. C. Lockemann (Hrsg.), J. W. Schmidt (Hrsg.), „Datenbank-Handbuch“, Springer-Verlag, Berlin, 1993

[LuSt93] G. F. Luger, W. A. Stubblefield, „Artificial intelligence: structures and strategies for complex problem solving“, Benjamin/Cummings Publishing, Redwood City, California, 1993

[MaPe96] F. Maurer, G. Pews, „Supporting Cooperative Work in Urban Land-Use Planning“, Proc. COOP-96, 1996

[Maur96] F. Maurer, „Project Coordination in Design Processes“, Proc. of WET ICE ’96, IEEE press, 1996

[Meta97a] Metaphase Enterprise 3, „Metaphase Object Management Framework Administrator’s Manual“, Publication Number MT00205D, Structural Dynamics Research Corporation, Februar 1997

[Meta97b] Metaphase Enterprise 3, „Metaphase Object Management Framework User’s Manual“, Publication Number MT00206D, Structural Dynamics Research Corporation, Februar 1997

[Meta97c] Metaphase Enterprise 3, „Metaphase Integrator Toolkit Customization Manual“, Publication Number MT00221D, Structural Dynamics Research Corporation, Februar 1997

[Meta97d] Metaphase Enterprise 3, „Metaphase Model Reference“, Publication Number MT00223D, Structural Dynamics Research Corporation, Februar 1997

[Meta97e] Metaphase Enterprise 3, „Metaphase Integrator Toolkit Application Programmer’s Interface Reference“, Publication Number MT00224D, Structural Dynamics Research Corporation, Februar 1997

[Meta97f] Metaphase Enterprise 3, „Metaphase Installation Guide for Unix and WindowsNT“, Publication Number MT00300A, Structural Dynamics Research Corporation, Oktober 1997

[Meta97g] Metaphase Enterprise 3, „Integrator Toolkit Application Programmer’s Interface Supplement“, Publication Number SDO-103A, Structural Dynamics Research Corporation, Oktober 1997

[NaWe94] M. Nagl, B. Westfechtel, „A Universal Component for the Administration in Distributed an Integrated Development Environments“, Technical Report 94-08, RWTH Aachen, 1994

[Petr93] C. J. Petrie, „The Redux’ Server“, Proc. International Conference on Intelligent and Cooperative Information Systems (ICICIS), Rotterdam 1993

[PWD97] B. Prasad, F. Wang, J. Deng, „Towards a Computer-Supported Cooperative Environment for Concurrent Engineering“, Technomic Publishing, Volume 5, Number 3, 1997

[ReDa97] M. Reichert, P. Dadam, „A Framework for Dynamic Changes in Workflow Management Systems“, Proc. 8th Int. Workshop on Database an Expert Systems Applications, IEEE press, Toulouse 1997

[ReDa98] M. Reichert, P. Dadam, „ADEPT$_{flex}$ - Supporting Dynamic Changes of Workflows Without Losing Control“, Journal of Intelligent Information Systems 10, Kluwer Academic Publishers, 1998

[RiMi97] N. Ritter, B. Mitschang, „Die Assistenzfunktion kooperativer Designflows - verdeutlicht am Beispiel von CONCORD“, Informatik Forsch. Entw., Springer-Verlag 1997

[RMH+94] N. Ritter, B. Mitschang, T. Härder, M. Gesmann, H. Schöning, „Capturing Design Dynamics - The CONCORD Approach“, Proc. 10th Int. Conference on Data Engineering, IEEE press, Houston 1994

[RMHN95] N. Ritter, B. Mitschang, T. Härder, U. Nink, „Unterstützung der Ablaufsteuerung in Entwurfsumgebungen durch Versionierung und Konfigurierung“, 1995

[SWZ95] A. Schürr, A. Winter, A. Zündorf, „Graph grammar engineering with PROGRES“, Proc. European Software Engineering Conference, Barcelona, 1995, Springer-Verlag

[Tane95] A. S. Tanenbaum, „Verteilte Betriebssysteme“, Prentice Hall, München 1995

[Verl97] R. Verling, „Die Gestaltung virtueller Unternehmen - Potentiale durch Informations- und Kommunikationssysteme“, Diplomarbeit, Universität Zürich, 1997

[Vers95] G. Versteegen, „Alles im Fluß - Die Ansätze der Workflow Management Coalition“, iX 3/95, Heinz Heise Verlag, Hannover 1995

[Walt98] T. Walter, „Visualisierungsmethoden bei Workflow-Management - Prototyping und Showcases“, D-CSCW ’98, B. G. Teubner, Stuttgart 1998

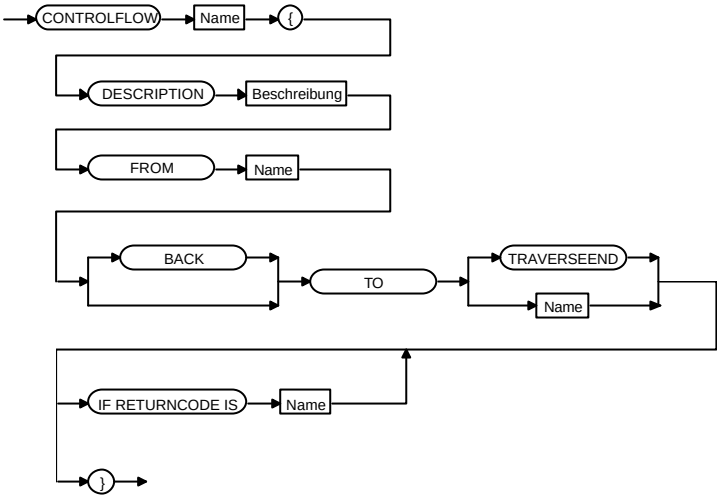
[WfMC94] Workflow Management Coalition, „The Workflow Reference Model“, Version 1.1, Document Number TC00-I003, Brüssel 1994

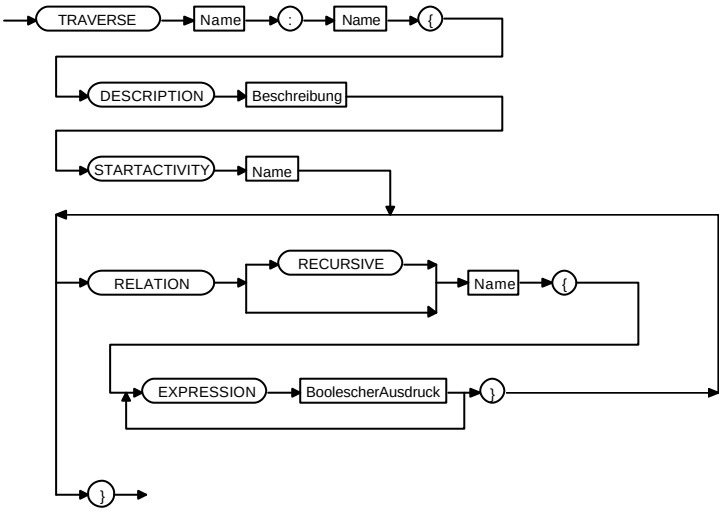
[WfMC96] Workflow Management Coalition, „Terminology & Glossary“, Version 2.0, Document Number WFMC-TC-1011, Brüssel 1996

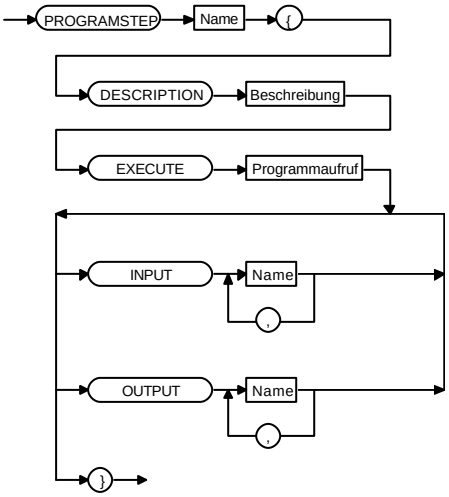
[WPS+97] M. Weber, G. Partsch, A. Scheller-Huoy, J. Schweitzer, G. Schneider, „Flexible Real-time Meeting Support for Workflow Management Systems“, Proc. 30th Int. Conference on System Sciences, IEEE press, Maui 1997

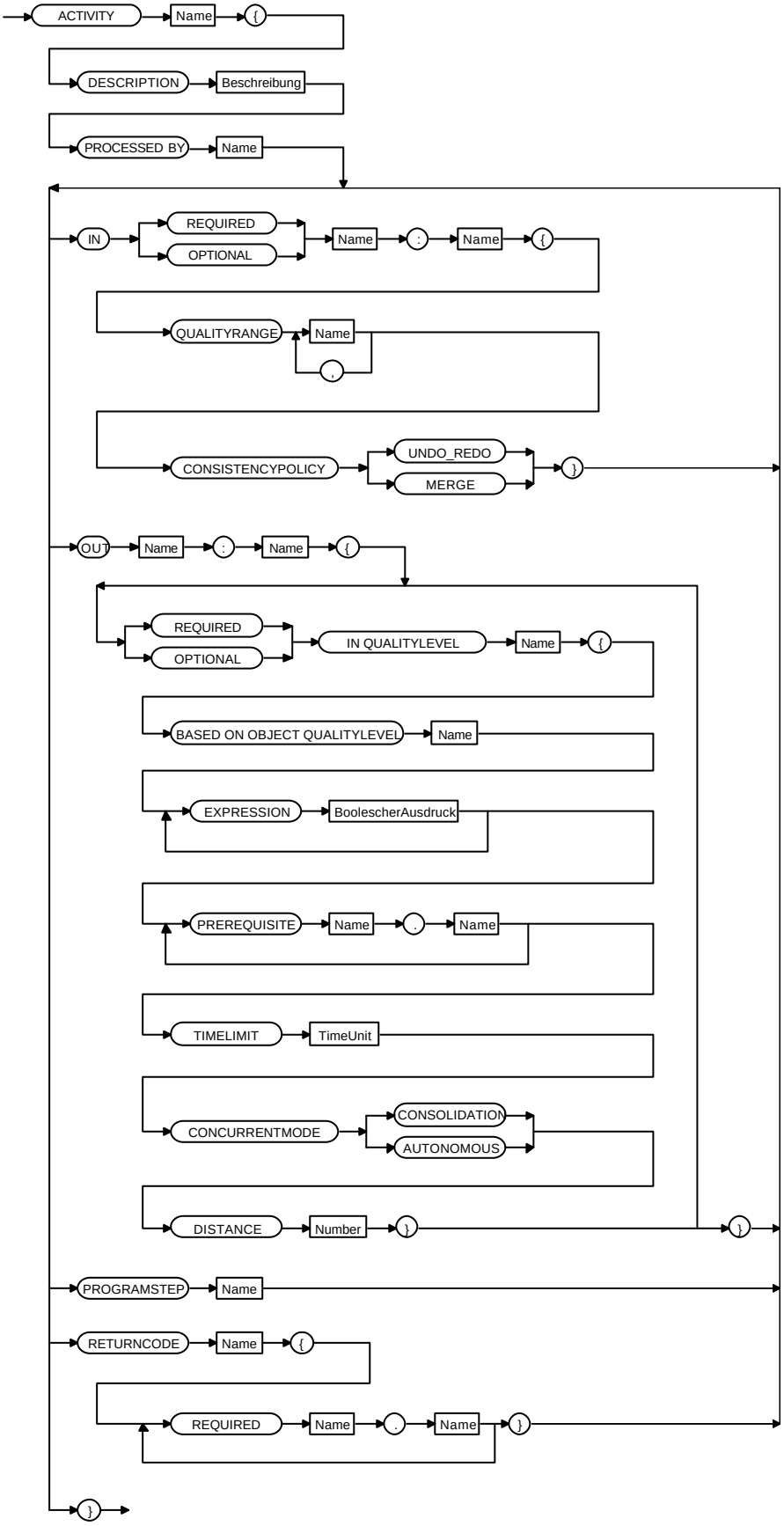
[WPS98] M. Weber, G. Partsch, A. Scheller-Huoy, „WoTel - Workflow und Tele-kooperation“, http://www-vs.informatik.uni-ulm.de/projekte/WoTel/wotel_de.html

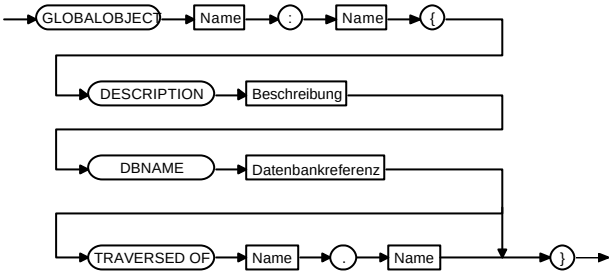
Anhang: Syntaxdiagramme

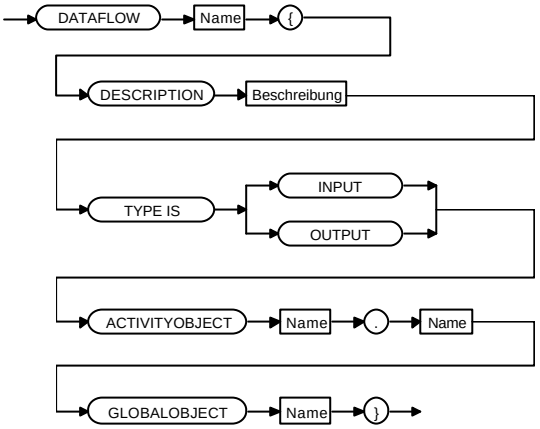


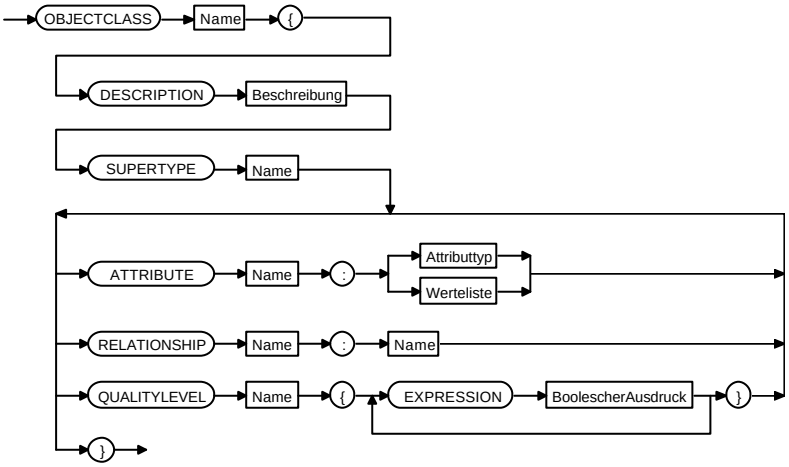


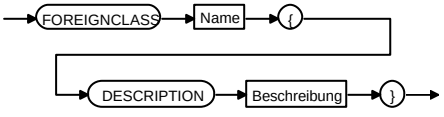


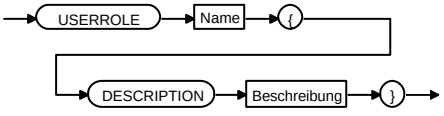

ACTIVITY
Name
{
DESCRIPTION
Beschreibung
PROCESSED BY
Name
IN
REQUIRED
OPTIONAL
Name
:
Name
{
QUALITYRANGE
Name
,
CONSISTENCYPOLICY
UNDO_REDO
MERGE
}
OUT
Name
:
Name
{
REQUIRED
OPTIONAL
IN QUALITYLEVEL
Name
{
BASED ON OBJECT QUALITYLEVEL
Name
EXPRESSION
BoolescherAusdruck
PREREQUISITE
Name
.
Name
TIMELIMIT
TimeUnit
CONCURRENTMODE
CONSOLIDATION
AUTONOMOUS
DISTANCE
Number
}
}
PROGRAMSTEP
Name
RETURNCODE
Name
{
REQUIRED
Name
.
Name
}
}


GLOBALOBJECT
Name
:
Name
{
DESCRIPTION
Beschreibung
DBNAME
Datenbankreferenz
TRAVERSED OF
Name
.
Name
}


DATAFLOW
Name
{
DESCRIPTION
Beschreibung
TYPE IS
INPUT
OUTPUT
ACTIVITYOBJECT
Name
.
Name
GLOBALOBJECT
Name
}


OBJECTCLASS
Name
{
DESCRIPTION
Beschreibung
SUPERTYPE
Name
ATTRIBUTE
Name
:
Attributtyp
Werteliste
RELATIONSHIP
Name
:
Name
QUALITYLEVEL
Name
{
EXPRESSION
BoolescherAusdruck
}
}


FOREIGNCLASS
Name
{
DESCRIPTION
Beschreibung
}


USERROLE
Name
{
DESCRIPTION
Beschreibung
}

Erklärung

Name: **Ralf Sauter** Matr.-Nr.: **0235325**

Ich erkläre, daß ich die Diplomarbeit selbständig verfaßt und keine anderen als die angegebenen Quellen verwendet habe.

Ulm, den ....

(Unterschrift)